André Fonseca dos Santos Dias Vieira

Licenciatura Engenharia Informática

Context-Aware Personalization Environment for Mobile Computing

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática

Orientador : Nuno Manuel Robalo Correia, Prof. Catedrático, Faculdade de Ciências e Tecnologia - Universidade Nova de Lisboa

Júri:

Presidente:	Doutor José Augusto Legatheaux Martins
Arguente:	Doutora Ana Paula Pereira Afonso
Vogal:	Doutor Nuno Manuel Robalo Correia

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Dezembro, 2012

Context-Aware Personalization Environment for Mobile Computing

Acknowledgements

Usually, in this section, most colleagues write their gratitude towards loved ones, friends, family and advisors. I follow that approach and also thank all of them for their support for the past year, which was crucial for the success of this work. Still, above them all, I want to focus my gratitude on a very special person that had a huge influence on the success of this work. Without him none of this would ever be possible. Thank you very much, to myself!

Abstract

Currently, we live in a world where the amount of on-line information vastly outstrips any individual's capability to survey it. Filtering that information in order to obtain only useful and interesting information is a solution to this problem.

The mobile computing area proposes to integrate computation in users' daily activities in an unobtrusive way, in order to guarantee an improvement in their experience and quality of life. Furthermore, it is crucial to develop smaller and more intelligent devices to achieve this area's goals, such as mobility and energy savings. This computing area reinforces the necessity to filter information towards personalization due to its human-centred paradigm.

In order to attend to this personalization necessity, it is desired to have a solution that is able to learn the users preferences and needs, resulting in the generation of profiles that represent each style of interaction between a user and an application's resources (e.g. buttons and menus). Those profiles can be obtained by using machine learning algorithms that use data derived from the user interaction with the application, combined with context data and explicit user preferences.

This work proposes an environment with a generic context-aware personalization model and a machine learning module. It is provided the possibility to personalize an application, based on user profiles obtained from data, collected from implicit and explicit user interaction. Using a provided personalization API (Application Programming Interface) and other configuration modules, the environment was tested on LEY (Less energy Empowers You), a persuasive mobile-based serious game to help people understand domestic energy usage.

Keywords: Generic Personalization Model, User Profiles, Mobile Computing, Context-aware Personalization, Personalization of Applications

Resumo

Atualmente, a quantidade de informação on-line supera a capacidade dos utilizadores a assimilarem. Filtrar essa informação de modo a obter conteúdo útil e relevante é uma solução para o problema.

A computação móvel propõe integrar computação nas atividades diárias dos utilizadores de forma não obtrusiva, com o intuito de garantir uma melhoria na sua experiência e qualidade de vida. É crucial desenvolver dispositivos mais pequenos e inteligentes para atingir os objectivos de poupança energética e mobilidade. Dadas estas limitações tecnológicas, é reforçada a necessidade de filtrar informação em direção à personalização, devido ao seu paradigma centrado no utilizador.

Deseja-se uma solução que considere as necessidades e preferências de cada utilizador, de modo a gerar perfis representativos de cada estilo de interação entre o utilizador e recursos de uma aplicação (e.g. botões e menus). Estes perfis podem ser obtidos com a utilização de algoritmos de aprendizagem automática que usem dados advindos das interações do utilizador.

É proposto um ambiente baseado num modelo genérico de personalização sensível a contexto. Tem como objetivo personalizar uma aplicação de forma não obtrusiva, com base em perfis de utilizador. Estes perfis são gerados por algoritmos de aprendizagem automática que usam dados recolhidos através da interação explícita e implícita entre o utilizador e a aplicação. Usando uma interface de programação e outros módulos de configuração, o ambiente de personalização foi testado na aplicação LEY (Less energy Empowers You), um jogo sério persuasivo para dispositivos móveis, cujo intuito é consciencializar as pessoas acerca do seu consumo energético doméstico.

Palavras-chave: Modelo de Personalização Genérico, Perfis de Utilizador, Computação Móvel, Personalização de Aplicações, Personalização Sensível a Contexto.

Contents

List of Figures

List of Tables

Listings

1

Introduction

About two decades ago, the World Wide Web started to pervade almost every aspect of our daily life enabling us to access information through a distance of thousands of kilometres in less than a second. Initially, the amount of transferred data was rather small in comparison to what we see currently, but due to a quick technological evolution, systems greatly improved their capabilities, granting to its users an increase in performance and in storage devices capacity. Given this situation, the amount of transferred data between different entities also increased at the same rate which resulted in a drastic increment of information.

This increase in machine resources also gave us the technology to create computer systems of several sizes and formats, resulting in the integration of those machines in several everyday situations. Computing is no longer just about using a desktop computer. Tablets, laptops, mobile phones and other gadgets are examples of how technology has evolved. Digital devices are getting smaller, faster and more efficient which, given the circumstances, explains the emergence of fields like pervasive and mobile computing.

While desktop computing presents a static nature, mobile computing is more prone to location variance and therefore, mostly used by moving users [PAS06]. Due to its mobility, users can use their mobile devices anywhere and anytime, which makes the device more personal than a normal desktop workstation. This difference results in a lack of adequacy of typical desktop solutions, when applied to mobile computing, which is why those solutions can benefit with an increase in personalization. Mobile computing also presents some limitations in comparison to desktop computing such as screen size limitations, lower computing power and necessity for energy savings. These restrictions may be soothed with more intelligent and efficient software, which are two points that can be provided with the integration of personalization on those devices.

1.1 Motivation

Pervasive computing demands adaptation and personalization as its basis, in order to offer the right personalized information to the user. To achieve this objective there are several issues that need some attention, including user preferences, heterogeneous environments and devices, dynamic user behaviour and user privacy [Mad12]. Concerning mobile computing, it is important that the system only takes into account the most accurate pieces of information so that the application may fit the users' needs and interests. This allows:

- The capability to satisfy the needs of every user independently of age and skills. Less skilled users probably require a simpler interface, in comparison to expert users that have a more detailed oriented approach.
- The appearance of dynamic interfaces that fit the users' usage habits. For instance, many applications allow the users to execute the same function in different ways, or if a user wants to close a window, most applications allow a keyboard shortcut or GUI shortcuts.
- Dynamic personalization that varies according to the context of usage. For example, the user might want to interact with certain applications in different ways if s/he is at work, home or other places.

It is crucial for the application to learn about the users' preferences and interests in order to provide good personalization. Given the need to grant personalization, machine learning techniques such as clustering and classification can be useful to accomplish good results, because they provide the capability to build models of each user, that represent their application profile.

Lying hidden in data there is precious information that is potentially useful and is rarely made explicit or used. This information can give, for instance, precious hints of user behaviour. Every click, every submission, every request can be turned into data and, by analysing it, specific patterns arise. Given the potential usefulness of those patterns it is desirable for computers to use them as extra information in their computation. Machine learning is a type of artificial intelligence that provides computers with the ability to learn without being explicitly programmed. They permit to extract relevant information from data generated from the usage of applications by their users. Those techniques allow to know more about each user and perceive what users like, what they are interested in, and also their capabilities or skills. All this user information can be used to adapt and personalize applications so that information can be filtered to what is needed and desired by each user.

However, there is a limitation in this approach. Users are human-beings and not deterministic machines that behave always in the same way. Our decisions are affected by other variables such as mood, company of others, and in the case of mobile computing

other variables such as location and weather conditions. That is why context information is extremely important to fine-tune the process of pattern extraction and user profile generation, and therefore deliver a more accurate personalization.

Unfortunately, there are still some limitations in this process. Looking at the current state of art related to context-aware systems (systems that take the context element into account), it is possible to perceive that most implementations are domain-driven, i.e., they are developed taking into account their specific area, such as cinema or tourism. The problem with this issue is that it does not provide much flexibility, which means that if someone wants to develop a system that profiles its users, the development would have to be done almost from scratch. This work's objective is to overcome this limitation.

1.2 Description and Context

This dissertation work was done in the scope of the DEAP (Developing Environmental Awareness with Persuasive Systems) project, which is being developed by CITI (Centro de Informática e Tecnologias de Informação) from FCT-UNL (Faculdade de Ciências e Tecnologia - Universidade Nova de Lisboa) in collaboration with UE (Universidade de Évora) and partly funded by FCT/MCTES (Fundação para a Ciência e Tecnologia) with the grant PTDC/AAC-AMB/104834/2008. The DEAP project introduces a new paradigm for environmental awareness, which will help motivating citizens to become more environmentally responsible in their everyday life, engaging them in environmental preservation activities. The purpose of this project is the study of how to stimulate citizens' responsible environmental behaviour changes through interactive public ambient displays that sense and react according to users' activities.

As a case study, this dissertation was applied and tested with LEY (Less energy Empowers You), a sub-project of DEAP, which is a persuasive mobile *serious* game approach to help people understand domestic energy usage in order to change their negative habits. The mechanics of the game are based on real-time domestic energy usage monitoring. The application is directed to households, but it should permit an easy extension to schools and offices scenarios. LEY benefits from the usage of different types of personalization, which is used to adapt LEY to different types of users with different needs and desires.

1.3 Presented Solution

In order to avoid the issues mentioned previously, the presented solution consists of an environment that integrates three modules: 1. A framework composed by a generic model, that offers an API (Application Programming Interface) to provide personalization to client applications; 2. An XML registration service which aims to configure applications within the environment; 3. A library of machine learning algorithms. This environment is referred as CAPE (Context-Aware Personalization Environment).

Personalization is provided based on a machine learning module that relies on implicit interaction stream data, context of usage data and explicit user preferences. This module aims to ease and automate the implementation of personalized mobile systems. Clustering as an unsupervised learning approach is used because CAPE aims to provide personalization with a minimal effort by the developer. If a traditional classification approach was to be used, due to its supervised learning nature, it would be necessary to have for each application a collection of labelled data to train the system. The problem is that in the beginning of each application's life there will never exist any data associated to it. Therefore, that initial data would have to be artificially created and estimated by the developer, which is not convenient or even effective in many circumstances. This means that the usage of clustering makes the personalization process much simpler and flexible for the application developer.

CAPE is used in the form of personalization APIs and configuration modules, which give the developer a high-level development platform because the algorithms are already implemented and only need to be combined and configured according to specific project's needs. Application developers are the main target of this solution. They analyse and define the data that is possible to extract from the interaction between user and application; create relations between different types of data to make it more expressive; define which personalization options are offered to each user; create combinations of user profiles that define each personalization option.

1.4 Contributions

The main contributions are:

- A personalization environment which offers an API composed of several methods: Creation of new users in the data model; update of a user's interaction stream data; update of a user's preferences data; request of a personalization option that will be used in on the client side to offer personalization.

- The integration of personalization on LEY mobile application using CAPE. Users' interaction stream data, the corresponding context of access and users' preferences are used to generate user profiles and personalize some application's elements such as the interface and level of alerts the application sends to each user.

- A study that evaluates two important points in this work: the developed personalization data model and the context segmentation approach. The adopted personalization data model is evaluated in order to know its usefulness, expressiveness and complexity. The adoption of context segmentation as a technique to refine the personalization results is also studied, as a mean to evaluate how interesting and effective is its usage.

- Writing and submission of scientific articles regarding this work's theme. At the moment of this delivery, a work in progress paper was accepted and presented at the ACM Ubicomp 2012 conference [1] [MVC12].
- Participation in *Fraunhofer Portugal Challenge 2012* contest [2]. The submission successfully reached the 2nd phase of evaluation.

1.5 Document Structure

This document is structured in five main chapters described below:

- **Chapter 1 - Introduction:** Presents a global vision of this work, concerning motivation, description and context, the proposed solution and the main contributions.
- **Chapter 2 - Related Work:** Establishes a connection between the objectives of this work and the related work. It starts by giving an introduction about machine learning, overviews recommender systems and clustering techniques, makes an analysis of how to integrate context in a machine learning approach, includes an analysis and comparison of two machine learning frameworks and external contextual tools, and also presents some projects that influenced this work.
- **Chapter 3 - CAPE:** Explains the core of this work. Starts by broadly explaining the theory behind the presented idea, the requirements to use the framework, the explanation of the personalization data model, the environment's architecture and finally the work flow of the provided algorithms.
- **Chapter 4 - Case study - LEY:** Reports how CAPE was used to personalize LEY software and what kind of personalization was achieved.
- **Chapter 5 - Evaluation:** Presents the results and conclusions taken from the tests that were applied to surveyed software developers, in order to collect more information about CAPE.
- **Chapter 6 - Conclusions and Future Work:** This Chapter mainly contains the final conclusions of this work and ideas for the future.

[1] http://www.ubicomp.org/ubicomp2012/
[2] http://www.challenge.fraunhofer.pt/

2

Related Work

This chapter contains information related to the techniques and technologies required to develop the given solution, also presenting relevant related projects.

The first section gives an introduction to the main concepts of machine learning, which is followed by a brief overview of recommender systems technology that was previously considered to be used, and finally addresses clustering technology as a solution for what is intended by this work. The second section introduces context as a complement to the machine learning solution. The third section makes an analysis of two machine learning frameworks which were considered for this work and also an overview of weather external services, which add additional context information, that can prove to be useful in the generation of user profiles. The final section cites some relevant research projects that influenced this work.

2.1 Machine Learning

It is legitimate to say that computers were created to make calculations and solve problems. Most of the time those problems can be surpassed using a specific algorithm that given some input will deliver the desired output with the solution. It happens that there are problems which cannot be solved with a simple algorithm. For instance, an electronic mail spam detector cannot be implemented in a simple way, because spam changes in time and from individual to individual. This is a clear case of lack of knowledge by the application, but on the other hand most of the time there are large amounts of available data that indicate what the user considers as spam or regular mail. This data can be used to "teach" a computer program to differentiate what is spam from what is regular mail. In practice the algorithm is generated automatically after the analysis task [Alp04]. This

is just a very limited example for a machine learning application, because the area is very wide and involves several different technologies adapted to problems with different domains. [Zha10] organizes machine learning algorithms into a taxonomy, based on the outcome of the algorithm:

- **Supervised learning** - Consists on the generation of a function that maps input to the desired output, using training data. The training data consists of a set of training examples where each example is composed by a pair where the first element is an input object (for instance, a vector of values) and the second element is the output label. The algorithm analyses the training data and produces an inferred function called *classifier* or *regression function*, depending on the type of output;
- **Unsupervised learning** - Tries to find a structure in unlabelled data, i.e., the examples do not have an output value/label which makes it impossible to have a sense of error. This is what distinguishes unsupervised learning from supervised learning;
- **Semi-supervised learning** - Combination of labelled and unlabelled examples to generate an appropriate function or classifier;
- **Reinforcement learning** - The algorithm learns a policy of how an *agent* should act given an observation of the *world/environment*. Every action has some impact in the environment, and the environment provides feedback that guides the learning algorithm. Reinforcement learning differs from standard supervised learning in that correct input/output pairs are never presented, nor sub-optimal actions explicitly corrected;
- **Transduction** - Also known as *transductive inference*, is quite similar to supervised learning, but does not explicitly construct a function. It tries to predict new outputs based on training inputs, training outputs and also new inputs. That is the opposite of induction, which reasons from observed training cases to general rules, that are will be applied to test cases;
- **Learning to learn** - Also known as *multi-task learning*, is an approach where the algorithm learns a problem with other related problems at the same time, based on previous experience. Due to this collaboration, it provides learning that uses the commonality among the task, leading to a better model for the main task.

2.1.1 Recommender Systems

In the early stages of this work, recommender systems were considered as a candidate machine learning application, able to provide personalization to mobile applications. Chapter 1 states some mobile computation limitations such as screen size and sub-par processing capabilities. These limitations can be avoided if information is limited to what is strictly essential and useful for each user.

Burke in [Bur02] defines Recommender Systems as a subclass of information filtering system, which are used to describe any system that produces individualized recommendations as output, or has the effect of guiding the user in a personalized way to interesting or useful objects in a large space of possible actions. They are primarily directed towards individuals who lack sufficient personal experience or competence to evaluate the potentially large number of alternative items that a source may offer.

RSs should deliver a personalized output, because it is useful for different users or user groups to receive different recommendations adapted to their interests. Therefore, it is common for this kind of systems to collect data from users and create models out of it, in order to predict what are the most suitable items, based on preferences or constraints. Many RSs have the ability to learn from the provided data in order to predict user evaluations for items, or correctly rank items for a user. That is why they benefit from the results of several computer science fields such as machine learning, data-mining and human-computer interaction [RRSK11].

For this work's purposes it would be ideal to use recommender systems to collect information about each user's habits, interests and needs, so that a mobile application could filter irrelevant information, and then provide personalization to each user. The problem with this approach resides in the fact that most traditional recommender systems use an abstraction *user-item*, i.e., they seek to predict the level of preference that a user would give to an item, they had not yet considered. Since this work is supposed to result in a generic personalization environment, this abstraction proves to be a severe limitation because in most applications the notion of item is not existent. It would be extremely complex to virtualize items according to each user's interaction stream with the application, which would result in a even more difficult personalization of that application. For these reasons, recommender systems do not seem to be an intuitive and helpful approach for a personalization environment that aims to be generic enough and adaptable to each possible type of application.

2.1.2 Clustering

As referred in chapter 1, this work focuses on the personalization of applications. The kind of data that is possible to pervasively derive from regular applications is not labelled, i.e., it is usually raw data like the number of clicks or the number of accesses to applications' resources. Therefore, based on the previous list, we can point out unsupervised learning as a technology with potential to achieve the desired purposes.

Clustering or *Cluster analysis* belongs to the unsupervised machine learning family, and consists on the organization of a collection of patterns (usually represented as a vector of measurements, or a point in a multidimensional space) into clusters based on similarity, i.e. it is a dividing of data into groups of similar objects [Ber02] [JMF99]. Patterns within a valid cluster are more similar to each other than they are to a pattern belonging to a different cluster.

Clustering is not a specific algorithm by itself, but the name given to the general task to be solved. It plays a big role in data mining applications such as scientific data exploration, information retrieval and text mining, spatial database applications, web analysis, marketing, medical diagnostics and computational biology.

The previous clustering definition permits to assume that the objects to be clustered can be represented as points in the measurement space. It is easy to recognize clusters when looking at a collection of points represented in two dimensions. While it is easy to give a functional definition of a cluster, it is very difficult to give it an operational definition. Objects can be grouped into clusters with different purposes in mind. In fact, data can reveal clusters of differing "shapes" and "sizes". For example, figure 2.1 illustrates a collection of points in a two-dimensional canvas. The amount of clusters in the figure can differ from person to person. At the global/higher level of similarity, we perceive two clusters in this collection, but at the local level we perceive 6 clusters. None of the answers is correct because it depends on multiple scales of visualization. Thus, the crucial problem of identifying clusters in data is to specify what proximity is and how to measure it [JD88].

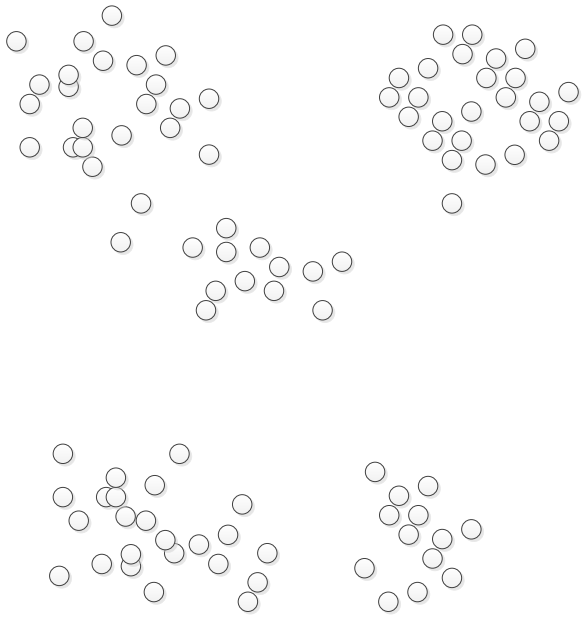


Figure 2.1: Several points spread out

Clustering a collection of items usually involves three entities:

- **An algorithm** - The method that is used to group the data together into groups of similar objects;
- **A notion of both similarity and dissimilarity** - In order to organize data into patterns, data must be compared between each other to predict a level of similarity or dissimilarity;

- **A stopping condition** - Eventually the algorithm will have to stop. That should happen when it is not possible to move objects around between clusters, i.e. when each cluster is already quite dissimilar from the others.

One way to imagine clusters is by picturing circles around points/objects. Figure 2.2 shows the clustering of points in a standard two dimensional Cartesian plane. Each circle represents one cluster containing several points. This circle representation also means that each cluster will have a radius and centre point to describe them. The circle centre is called the *centroid*, or *mean* of the cluster. Of course there are clustering techniques that are not compatible with this type of representation, but this is a good way to visualize a cluster and is generally adopted [OADF11].

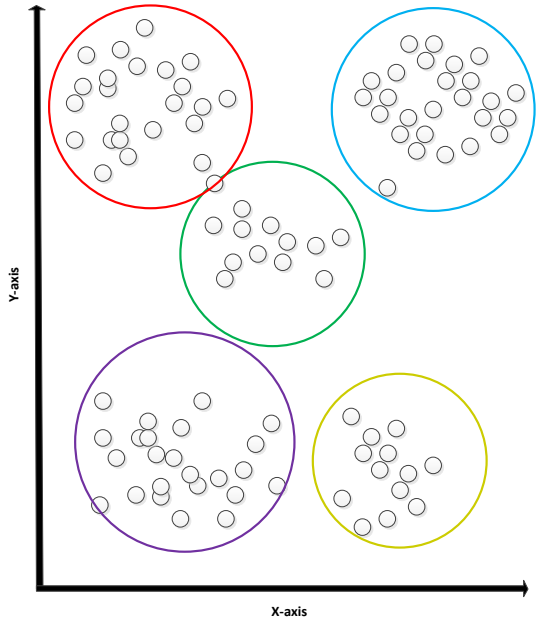


Figure 2.2: Several points grouped in clusters

2.1.3 Distance Measures

The notion of distance between points is crucial to the success of the clustering operation, and that is why the most important issue in clustering is finding a function that quantifies the distance between any two data points as a number. There is a large collection of distance measures available. It is not possible to identify one distance measure as the best of all, because the performance and effectiveness of each one strongly depends on the used data. Some examples of distance measures [OADF11] [TSK05]:

- **Euclidean distance measure** - Given by the Pythagorean formula, it represents the normal distance between two points that one can measured by a ruler.

- **Squared Euclidean distance measure** - It is the square of the value returned by the Euclidean distance measure. Used in order to place progressively greater weight on objects that are further apart.

- **Manhattan distance measure** - This distance measure takes its name from the grid-like layout of streets in Manhattan, New York. It is the simple sum of the horizontal and vertical components, whereas the diagonal distance might be computed by applying the Pythagorean theorem. It proves useful in domains which do not allow diagonal movement, such as Chess games.

- **Cosine distance measure** - The cosine distance measure requires to think of points as vectors from the origin to those points, so that the vectors form an angle, between them. When the angle is small the vectors must be pointing in somewhat the same direction, which makes the points close to each other. That value is represented by the angle's cosine, which means the length of the two vectors is not taken into account. It is particularly useful to compare documents in text mining.

- **Tanimoto distance measure** - As referred previously, cosine distance measure disregards the lengths of vector and that can be a problem in some situations. If we consider points P1(1.0, 1.0), P2(4.5, 4.5) and P3(5.0, 5.0), cosine distance would not be able to capture the fact that P2 and P3 are closer in comparison to P1. Euclidean distance measure would be able to reflect that, but would have no information regarding angles. This means that Tanimoto distance takes the best of both Euclidean and Cosine distance measures.

- **Chebyshev distance measure** - Also known as Maximum Value distance, is a metric defined on a vector space where the distance between two vectors is the greatest of their differences along any coordinate dimension. If we consider points P1(0, 1) and P2(1,7), the output of the distance measure is the greatest between |0-1| or |1-7|, i.e. 1 or 6, therefore the result would be obviously 6. Similarly to Manhattan distance, it is also useful in Chess-like domains.

- **Minkowski distance measure** - The Minkowski norm provides a concise, parametric distance function that generalizes many of the distance functions used in the literature. Its advantage is that mathematical results can be shown for a whole class of distance functions, and the user can adapt the distance function to suit the needs of the application by modifying the Minkowski parameter. Therefore, it can be considered as a generalization of both the Euclidean distance and the Manhattan distance [GKvR07].

2.1.4 Clustering methods

There is a large amount of clustering algorithms that vary significantly in their properties and, therefore, in the output of different results. It is not easy to categorize every clustering algorithm because the variety is very wide. Traditionally, clustering techniques are broadly divided in hierarchical, partitioning, density-based clustering and probabilistic clustering [Ber02].

2.1.4.1 Hierarchical Methods

Hierarchical clustering is a method of cluster analysis which seeks to build a hierarchy of clusters [OADF11] [Ber02]. It allows to group fine-grained clusters into bigger ones that are more generic. This can be done successively until a point where the clusters are so large and so generic that they are useless as groupings. Despite this problem, they are quite useful until a certain degree. Usually, there are two types of strategies for hierarchical clustering:

- **Agglomerative** - Each object starts in its own cluster and proceeds with a series of fusions of the n objects into groups as one moves up the hierarchy;
- **Divisive** - All objects start in one cluster, and splits are performed successively into finer groupings.

Hierarchical clustering allows embedded flexibility regarding the level of granularity of groups. It easily handles any forms of similarity and it is applicable to any attribute type, but it also has its own disadvantages such as vagueness of termination criteria and the fact that most algorithms do not review the clusters once constructed, with the purpose of improvement or refinement.

2.1.4.2 Partitioning Methods

When using a partitioning method the result is a set of M clusters, where each object belongs to one cluster. Each cluster has a *representative* that basically is sort of a summary description of all objects inside that cluster. The representative can assume many forms, depending on the data. For instance, if data is represented by real-valued data, the arithmetic average of the attribute vectors for all objects within a cluster can provide an appropriate representative.

There are several methods of partitioning clustering [RS10] [OADF11]:

- **K-means methods** - Aims to partition n points into k clusters in which, each point belongs to the cluster with the nearest *mean* (usually, weighted average). Obviously, this is not very helpful with categorical attributes, but has good geometric and statistical sense for numerical attributes. It starts with an initial set of k centroid points, that are chosen arbitrarily. An approximate way of estimating k is to guess based

on the data and the size of clusters to be used. The algorithm does multiple rounds of processing and refines the centroid locations until the criterion of maximum iterations is reached or until the centroids converge to a fixed point from which they do not move very much. The processing basically consists of two steps. The first one finds the points that are nearest to each centroid point and assigns them to that specific cluster. The second step recalculates the centroid point using the average of the coordinates of all the points in that cluster. These two steps are processed repeatedly until convergence is reached. Due to its simplicity K-means is a very fast algorithm, specially for a small number of clusters. It also produces tighter clusters then traditional hierarchical clustering, specially if the clusters assume globular geometry. K-means also has its limitations, specially when clusters strongly differ in size, density and have non-global shapes. It is also not very easy to compare the quality of the produced clusters;

- **Bisecting K-Means Method** - It is an extension of the K-means method. Beginning with a cloud of points, its global centroid w is computed. Then, a random point pL is selected among the cloud of points. With the pL point, a pR point is built as the symmetric of the pL point when compared to the main centroid w, i.e. the distance between pL and w is the same as the distance between pR and w. The collection of cloud points is separated in two: The ones closest to pL belong to sub-cloud L and the ones closest to pR belong to sub-cloud R. If there are more than 2 clusters, re-iterate for each sub-cloud R and L. The basic algorithm is very fast but poor in precision. Nevertheless, there are many optimizations for it;

- **Medoids Methods** - In contrast to the K-means algorithm, data-points are chosen as centres of clusters. Two advantages show up in comparison to the traditional K-means. First, it presents no limitations on attribute types, i.e. data does not have to be numerical, and second, the choice of medoids is dictated by the location of a predominant fraction of points inside a cluster, which makes it less sensitive to the presence of outliers (data-points far from the cluster medoid).

2.1.4.3 Density-based Algorithms

Density-based algorithms are capable of discovering clusters of arbitrary shapes. These algorithms group objects according to specific density objective functions. Density is defined as the number of objects in a particular collection/neighbourhood of data-points. This provides a natural protection against outlier data-points [JD88].

This type of clustering can assume two types [NM11]:

- **Density-Based Connectivity Clustering** - In this type, density and connectivity of clusters are both measured in terms of local distribution of nearest neighbours. All the points reachable from core objects (cluster internal points) can be factorized into maximal connected components serving as clusters. Those points that are not

connected to any core point are declared to be outliers, making them not covered by any cluster. The non-core points inside a cluster represent its boundary (different from outliers, because they belong to the cluster). DBSCAN and DBCLASD are two examples of density-based connectivity clustering methods, because they are related to training data points;

- **Density Functions Clustering** - In this type, density is computed by special density functions. The overall density is modelled as the sum of the density functions of all objects. Clusters are determined by *density attractors*, which are local maxima of the overall density function. DENCLUE is an example of density functions clustering.

2.1.4.4 Probabilistic Clustering

Probabilistic clustering is based on probabilistic modelling [Ber02] [TSK01] [XEKS98]. A probabilistic model is usually a characteristic shape or a type of probabilistic distribution of a set of points in a *n*-dimensional plane. There are several probabilistic models that fit known data patterns. Probabilistic clustering algorithms try to fit probabilistic models over a data set and try to make them fit, by adjusting the models' parameters. Since this fit rarely happens, these algorithms give a percentage match or a probability value, which indicates how well the model fits the cluster.

2.1.4.5 Conclusions

Data clustering has proven to be an interesting approach to find similarities in data and put that data into different groups or partitions. Clustering partitions a data set into several groups such that the similarity within the group is greater in comparison to data among other groups. An overview of the most used techniques is presented, showing that there is no better technique, because each one fits better with different data sets, i.e., they are developed for particular situations. A successful clustering operation depends on several factors such as the data quality, the way data is arranged, the number of clusters to be used, data preparation (e.g., irrelevant attributes), proximity measures and handling of outliers.

2.2 Context

Traditional machine learning techniques generally anchor in pure content-based analysis and do not take context into account. As a consequence, those technique's results that are not tailored to users' current context. There are some application domains that can greatly benefit with the addition of data related to contextual information, so that data can be grouped under the conditioning of certain circumstances [DPDM09] [WUS09]. With context, a machine learning system perceives a user model as something more dynamic, which is also more realistic. A user in some occasions might behave in a certain way, and in other situations he/she can have totally different interests or needs. That

is why the incorporation of context may be relevant, having a significant effect in the prediction accuracy of users' profiles.

2.2.1 Notions and Modelling

Context is applied in a multitude of different areas such as linguistics, philosophy, and obviously computer science. Each one of these disciplines tend to have their own definition of the word *context* that adapts to the respective field domain. It happens that literature does not agree with a particular definition, and each author uses the *dictionary definition* (which is very generic) and adds some information related to their own research. For instance, Brown et al. includes the date, the season, and the temperature [BBC97]; Dey et al. also includes the user's emotional status [DAS01]. Despite this lack of agreement, it is possible to point out that since mobile computing is related to space and movement, the "location" attribute is shared among most definitions.

According to [WS07] and several other authors, context should be considered as a separate entity in the data model and not as part of the users and preferences descriptions, because there are some substantial differences. Table 2.1 presents those differences.

Table 2.1: User Profile vs. Context

User Profile	Context
• Rather static and somewhat longer lasting	• Highly dynamic and transient
• Stored in user profile	• Not stored permanently
• Implicitly observed or explicitly provided by user	• Observed only, never manually entered by user
• More simple/raw nature	• More complex structure depending on its nature

Context is not always useful in the machine learning process, because the system might be using irrelevant information that is not compatible with the system's domain. Palmisano et al. say there is a trade-off between transaction homogeneity and data sparsity, because the act of providing contextual information will reduce the amount of user transactions in a given context, despite the increase in accuracy [PTG08]. This results in fewer data points to fit the model, while the homogeneity of the transactions increases granting an improved prediction accuracy. This makes context identification crucial for the machine learning tool's success. The process may be done in a manual way, where appropriate personnel (e.g., market experts and system designers) elect relevant context entities, based on their expertise; or in an *automated* way, where some techniques are applied to the system in order to study the effect of different context elements on the outcome. In this kind of situations it is common to use machine learning, data mining or even statistics software.

2.2.2 Identifying Context Data

Broadly, there are three ways to identify contextual data [RRSK11]:

- **Explicitly** - Consists in directly querying the user or other sources of data about the current access status.
- **Implicitly** -Data is gathered in an implicit way, i.e., there is no direct interaction between the system's users or other types of relevant context data. A GPS locator in a mobile device regularly updating the user's current location is an example of gathering context data in an implicit way.
- **Inferring** - It is possible to infer context using data mining or statistical techniques. In order to be successful, the application needs a predictive model that can be trained with data related to the systems usage. For example, a television set might be able to predict who is using the remote control by analysing data such as the watched programs, number of times and the rate that the user changes channels.

2.2.3 Techniques of Context Incorporation

In general, context can be introduced in three different ways:

- **Contextual pre-filtering** - Context data is used to filter out unwanted data before it is used by the machine learning tool.
- **Contextual post-filtering** - Analogous to pre-filtering. Instead of filtering data at the beginning, it is filtered in the end after executing the machine learning tool.
- **Contextual modelling** - Contextual data is used directly in the modelling technique.

There is still a limited amount of work comparing these different techniques [RRSK11], which is the reason why it is not possible to reach meaningful conclusions about the strengths and weaknesses of each context incorporation technique, including approaches that combine more that one type of context incorporation technique. This happens because their performance is strongly related to the application, domain, users and environment. Therefore, in order to achieve the best results, the three techniques should be implemented and tested to see which one(s) fits best for the application's purposes.

2.2.4 Flexibility

In the *Recommender Systems Handbook*, co-authors Adomavicius and Tuzhilin identify two properties that must be taken into account when using context, in order to achieve maximum flexibility and interaction [RRSK11].

One of them is related to complexity. Contextual information can assume different levels of complexity according to the nature of each context element. This fact has a significant influence in the system performance. For instance, a tourism domain application that suggests activities to the user has to take into account the weather, user personality, time, money and many other aspects, which results in a very complex process.

The other property concerns interactivity. Depending on the application, there might be the need to collect some context information that simply can not be inferred through the user's application's usage. Collecting that information can be quite intrusive and may not be easily provided by the users due to, for example, privacy issues.

2.3 Services and Software Analysis

This section covers the analysis of two machine learning frameworks that were considered for this work. It also presents a brief overview of some external services that could prove helpful to create more accurate data.

2.3.1 Apache Mahout

Mahout[1] is an open source machine learning library from Apache Software Foundation [OADF11]. It implements machine learning and collective intelligence algorithms, particularly recommender engines, clustering and classification. It is a framework written in the Java programming language that is intended to be used and adapted by developers. One of the strengths of Mahout is related to scalability, because it also aims to be a machine learning tool of choice when the collection of data to be processed is very large, perhaps too large for a single machine. Mahout uses, with some of its algorithms, the open source framework Apache Hadoop[2] in order to achieve scalable solutions. Hadoop is an open-source framework that supports data-intensive distributed applications. It enables applications to work with thousands of independent computers and petabytes of data, therefore reliability and scalability are two very important characteristics of it.

The first of the three machine learning approaches in Apache Mahout concerns recommendations. Recommender systems describe any system that produces individualized recommendations as output, or has the effect of guiding the user in a personalized way to interesting or useful objects in a large space of possible actions. They are primarily directed towards individuals who lack sufficient personal experience or competence to evaluate the potentially large number of alternative items that a source may offer [Bur02].

The second approach focuses on clustering techniques (see section 2.1.2).

Finally, the third approach is about classification. This technology aims to decide how much an element is part of some type or category. It helps to determine whether a new input or element matches a previously observed pattern or not. For instance it could be

[1] http://mahout.apache.org/
[2] http://hadoop.apache.org/

used to determine if a user is frustrated or satisfied about something, or if he would like red wine or white wine.

2.3.1.1 Clustering Techniques

As referred previously, this work uses clustering and that is why classification and recommendation will not be approached in this work. For clustering, Apache Mahout uses a *Vector* object to store data, which is basically an ordered list of values. In Mahout, vectors are implemented as three different classes, each of which is optimized for different scenarios: *DenseVector*, *RandomAccessSparseVector*, and *SequencialAccessSparseVector*. A Dense vector is an array of doubles, whose size is the number of features in the data, which means all the entries in the array are pre-allocated. A Random access sparse vector is implemented as a Map between an integer and a double, where only non-zero valued features are allocated. For last, a sequential access sparse vector is implemented as two parallel arrays, one of integers and the other of doubles. Just like with random access sparse vector, only non-zero valued entries are kept in it. The difference is that the first is optimized for random access while the latest is optimized for linear reading. For smaller examples this difference is irrelevant, but since Apache Mahout is designed to withstand large amounts of data this difference is very important. With these three implementations flexibility is provided to choose a vector class whose performance characteristics suit the nature of the data, the algorithm, and the way data is accessed by it [OADF11].

Regarding clustering techniques, Apache Mahout offers several different choices:

- **K-means clustering** - Mahout implements a basic K-means algorithm and also a version named MapReduce, which allows its usage in distributed computer networks for cases when data assumes a very large size, impossible to run in-memory. For more details concerning the basic K-means algorithm see section 2.1.4. Using the MapReduce approach, it is possible to split up the clustering algorithm to run on multiple machines, with each mapper getting a subset of the points. The mapper jobs will partially compute the nearest cluster by reading the input points in a stream-fashion. This MapReduce version of K-means is designed to run on an Apache Hadoop cluster making good use of its distributed computing capabilities.

- **Canopy clustering** - In many real-life clustering problems, the number of clusters is not known for certain which can result in loss of information. A class of techniques known as approximate clustering algorithms can estimate the number of clusters as well as the approximate location of the centroids from a given data set. Canopy clustering is one such algorithm. It is used to divide the input set of points into overlapping clusters known as canopies. This algorithm's main strength lies in its ability to create clusters extremely quickly (just a single pass over the data is required). The problem is that it may not give accurate and precise clusters, therefore its strength is also its weakness. Nevertheless, this algorithm can give the optimal

number of clusters without even specifying the number of *k* clusters, as required by k-means.

- **Fuzzy k-means clustering** - While k-means is an exclusive clustering algorithm, fuzzy k-means tries to generate overlapping clusters from the data set. Using an overlapping clustering approach, any point can belong to more than one cluster with a certain affinity value towards each. In a similar way to k-means, fuzzy k-means also has an in-memory implementation and a MapReduce implementation for distributed computing.

- **Dirichlet clustering** - For situations that demand hierarchical clustering in order to detect sub-clusters, but also require overlapping among the clusters, probabilistic clustering can be used (notice that hierarchical clustering algorithms do not support overlapping). Dirichlet clustering belongs to the probabilistic clustering family. While K-means and its derivatives, cluster points generated from an asymmetrical normal distribution, probabilistic clustering techniques test whether data fits a mathematical model (circular, oval, triangular, etc) by reading through the data vectors and calculating the probability of the model fitting the data (see section 2.1.4). Mahout's implementation of Dirichlet clustering goes beyond the traditional algorithm, by implementing it as a Bayesian clustering algorithm, which means that the algorithm can provide more than one explanation of the data. For instance, it can say that cluster 1 fits a circular model, cluster 2 fits an oval model and cluster 3 fits a triangular model. Just like K-means, Canopy and Fuzzy K-means, Dirichlet also has a MapReduce implementation for distributed computing.

Summary of each technique:

Table 2.2: Mahout's clustering techniques

Algorithms	**Fixed Clusters**	**Hierarchical**
K-means	Yes	No
Canopy	No	No
Fuzzy k-means	Yes	Yes
Dirichlet	No	Yes

2.3.1.2 Similarity Metrics

Similarity metrics are extremely important for the effectiveness of a clustering algorithm. The developers of Apache Mahout are aware of that because the framework offers several of them to its users: Euclidean distance measure, squares Euclidean distance measure, Manhattan distance measure, Cosine distance measure and Tanimoto distance measure. All of these are described in section 2.1.3. Apache Mahout also offers another distance measure that is not referred in that list, called *Weighted distance measure*. This is a generic

technique that allows the usage of Euclidean distance metric or Manhattan distance metric and assign a weight to each dimension in order to normalize them.

2.3.1.3 Conclusions

Apache Mahout is a growing open source framework with an active community that continuously contributes to its development. There are several tutorials available and a published "hands-on approach" book. It implements several clustering algorithms of different nature that accommodate several different types of data. There is also an option to integrate it with the Eclipse framework[3].

On the other hand, Apache Mahout has a dependency on Apache Hadoop project[4], which makes its execution quite slow for simple applications with a low amount of data, even with in-memory execution. This is due to all the overhead caused by Hadoop. Therefore it is possible to conclude that Apache Mahout can be quite useful for an implementation that uses a large amount of data and demands distributed computing.

2.3.2 Weka

The Weka workbench[5] is a collection of state-of-the-art machine learning algorithms and data preprocessing tools. It is designed to allow people to try out existing methods on new datasets in flexible ways. An extensive support for the whole process of experimental data mining is provided, in order to achieve its purpose. This includes the preparation of input data, the evaluation of learning schemes in a statistical way and also the visualization of the input data and the learning process result. The framework was developed at the University of Waikato in New Zealand and it is implemented in Java [WF05].

The workbench includes methods for the main data mining problems: regression, classification, clustering, association rule mining, and attribute selection. The focus will be given to clustering techniques.

2.3.2.1 Clustering Techniques

Before explaining every clustering technique that Weka has to offer it must be said that Weka requires the data to be submitted in a file in the formats .arff (Attribute-Relation File Format) or .csv (Comma-separated values).

Weka framework offers the following clustering techniques [SBL12]:

- **Cobweb algorithm** - The Cobweb algorithm yields a clustering dendrogram (used to represent hierarchical clustering) called *classification tree* that characterizes each cluster with a probabilistic description. Based on this probabilistic value, it allows the creation of classes (clusters) on the fly, because merging and splitting of those

[3] http://www.eclipse.org/
[4] http://hadoop.apache.org/
[5] http://www.cs.waikato.ac.nz/ml/weka/

classes are provided. It is also capable of doing a merge between two classes and after a while split that class in two, which means it is able to do bidirectional search, which is not possible to do using K-means algorithm. Regarding limitations, this algorithm does not consider attribute dependencies, and the probability distribution representation of clusters makes it quite expensive to update and store the clusters, specially when the attributes (points) have a large number of values. This happens because the time and space complexities depend on the number of attributes and also on the number of values for each attribute. Another problem has to do with the fact that the classification tree is not height-balanced which may result in a drastic degradation in time and space complexity when searching for specific values.

- **DBSCAN algorithm** - Density-based spatial clustering of applications with noise (DBSCAN) is a density-based clustering algorithm (see definition in section 2.1.4). The algorithm has several advantages: Unlike k-means, the number of clusters does not need to be known *a-priori*; it can find arbitrarily shaped clusters, including clusters that are completely surrounded by other clusters; has a notion of noise (irrelevant or meaningless data); only requires two parameters and is mostly insensitive to the way points are ordered in the database. It also has some disadvantages: It can only provide good clustering if its distance measure is in the function region Query(P, ϵ [6]), due to the so called "Curse of dimensionality" [7]; If there are large differences in densities, DBSCAN cannot cluster data sets well.

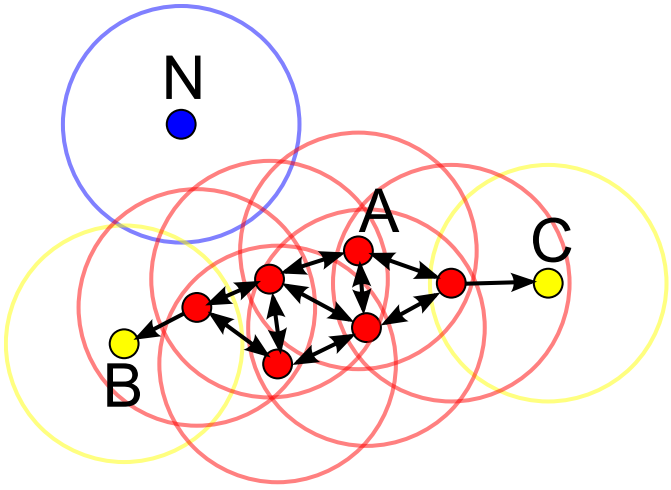


Figure 2.3: DBSCAN: Red points are *core points*, yellow points are *density reachable from A* and blue points are considered as noise

- **Expectation-maximization algorithm** - A distribution-based algorithm for finding

[6] (ϵ being the density-reachable distance)

[7] Euclidean distance has a serious limitation when comparing points with a different number of dimensions. Imagining that song *A* and song *B* have to be compared in order to know how similar they are. They have some genres in common but song *B* has a lot of genres that song *A* does not have. This yields an euclidean distance which, depending on the number of genres, can be larger than between two songs with no genre in common but less genres in total [NH12]

maximum likelihood, i.e., for estimating parameters in statistical models, where the model depends on unobserved latent variables (for instance, a medical diagnosis for patient *A* might have totally different variables in comparison to the diagnosis of patient *B*). Although this algorithm has a highly complex nature, it provides extremely useful results for real world data sets, and is also very useful when there is the need to perform a cluster analysis of a small region-of-interest, due to unsatisfying results obtained from k-means.

- **Farthest first algorithm** - A variant of k-means algorithm that places each cluster centre in turn, at the furthest point from the existing cluster centres. This greatly speeds up the clustering process in most cases since less reassignment and adjustment is needed. The results are close to optimal and it is suitable for large-scale data mining applications.

- **OPTICS algorithm** - The OPTICS (Ordering Points To Identify the Clustering Structure) algorithm is procedurally identical to the previously mentioned DBSCAN, but it addresses the problem of detecting meaningful clusters in data of varying density. It builds upon DBSCAN by introducing values that are stored with each data object. These values are referred as the *core distance*, i.e. the smallest ϵ value that makes a data object, a core object. There is also the *reachability distance*, which is a measure of distance between two given objects, and is calculated as the greater of either the core distance of the data object or the euclidean distance between the data object and another point. Both core distance and reachability distance are used to order the objects within the dataset.

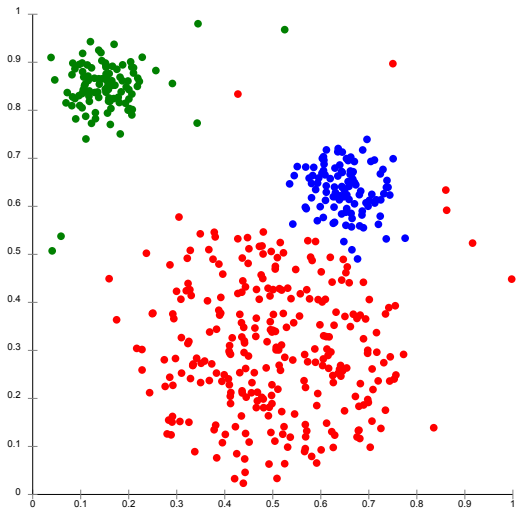


Figure 2.4: OPTICS handles different densities much better than DBSCAN

- **K-means clustering algorithms** - Already explained in section 2.1.4.

2.3.2.2 Similarity Metrics

Regarding similarity metrics, the Weka framework follows a different approach. Instead of providing a vast number of simple to complex distance metrics, it provides an interface named *DistanceFunction* that allows developers to easily implement their own distance measures. Unfortunately some clustering and classification algorithms are tightly implemented with their own default similarity metrics. For instance, the implementation of K-means algorithm is hard-coded with the Euclidean distance measure and the Manhattan distance measure, which makes it difficult for a developer to adapt the algorithm to its own distance metric. This means that in order to use a different distance measure, Weka users need to modify the source code of most algorithms to accommodate their own distance metrics. Still, Weka offers some distance measures such as Euclidean distance, Chebyshev distance, Manhattan distance, Minkowski distance and a function to normalize the dimensions of other distance metrics.

2.3.2.3 Conclusions

Weka is an open-source statistical and data mining workbench that has a lot of different types of machine learning algorithms. It is extremely easy to learn and use, and since it is written in Java it is platform independent. On the other end, it lacks variety in its collection of distance metrics, and the provided algorithms do not support distributed computation.

2.3.3 Contextual Tools

Section 2.2 says that context information can improve the results of some machine learning techniques. In order to incorporate context in those approaches, some auxiliary tools/services can prove to be helpful. In particular, the mobile computing area can greatly benefit from services that are related to mobility, such as weather stations.

World Weather Online[8] is a service that provides global weather forecast and weather content for its users. Furthermore, it covers two million worldwide cities and the forecast is trusted and used from small and medium enterprises to large corporate clients. It has a weather API, which allows developers and programmers to have free access to a five day weather content forecast in XML (Extensible Markup Language), CSV (Comma Separated Values), JSON (JavaScript Object Notation) data-interchange formats. It also allows using as input data the current location via zip-code, postcode or latitude and longitude. The free version offers up to 500 requests per hour which is enough to fulfil the testing requirements of this work.

[8] http://www.worldweatheronline.com/

Weather Underground[9] is another service that provides real-time weather information via the Internet. It is able to provide weather reports for cities across the world as well as local weather reports for newspapers and Web sites. It offers a variety of plans and pricing, though most use is free. Some example API methods include accessing weather information by cities and regions, retrieving forecast information, accessing satellite images, and current conditions by location. Contrasting from World Weather Online, Weather Underground does not require an API key, but the number of accesses is still tracked and limited. The free version offers 500 calls per day and 10 calls per hour.

These services might prove to be helpful to personalize applications according to the current local weather condition.

2.4 Related Projects

This section summarizes projects related to this work. The first sub-section analyses some projects concerning personalization of applications and the second sub-section refers to projects related to clustering as a machine learning approach.

2.4.1 Personalization of Applications

When multimodality became available, there was a lot of enthusiasm that was ultimately unjustified because people thought it allowed the usage of every application with their favourite modality while raising performance and user satisfaction at the same time [Kur07]. It was thought that personalization systems would enable this type of "automatic adaptation to user needs", but in reality a lot of different approaches were proposed with limited success and different results. Kurze offers an overarching personalization tool that acts independently from individual applications and that detects which modality or modalities are being used, deducting the reason *why* the user acted in the described way. This is a serious issue because multimodality offers the user a richer choice to select tools and methods to perform his task, resulting in an increase in input, which also increases the level of complexity in the user model. The most prominent key modules of the highly modular framework are the sensors which collect information on the user's interactions, the user model (profile database) which is implemented as centralized homogeneous storage and management unit and the machine learning unit (classification and recommendation modules). Initial observations showed that the framework produces valid and valuable results for the tested scenarios. Still, no proof was found that multimodal interfaces can benefit from the proposed personalization framework.

Concerning current developments toward the comparison of predictive algorithms for personalization of a mobile application, Nurmi et al. [NHL07] observed from their tests on different algorithms that those capable of inducing hierarchy between the contextual data attributes, work relatively well . This indicates that the collected contextual

[9] http://www.wunderground.com

data probably possesses a certain form of hierarchy among the relevant attributes, providing the prediction of users' behaviour under a certain context. It also describes a context-dependent user model that builds a predictive model for an individual user using data extracted from his/her past behaviour. It is said that although these kinds of models are suitable for situations where users tend to exhibit idiosyncratic behaviour, they also require large computational resources and a large amount of data.

In the future, advanced mobile services should make extensive use of personalization to improve their perceived value and quality [JTD06]. The study says that personalization shall be possible across a range of various devices, networks and services, which is why it is necessary to develop conceptual and abstract models that are independent of the underlying technologies. An ontology for personalization of information elements was modelled based on this conceptualization, using UML (Unified Modelling Language) notation. That representation of personalization information elements has a visual nature, which means that it is not machine processable. An OWL (Web Ontology Language) specification was developed in order to transform the visual model into a textual model, therefore making it machine processable.

Weiß et al. propose a generic framework for application developers, which allows content based filtering on users' preferences and involved context [WDFLP08]. The work introduces an approach for context-aware personalization of mobile multimedia services. Taking into account mobile requirements, it introduces different filtering methods that can appropriately be distributed among stationary server and mobile clients. Due to the fact that it was built for application developers, the framework is generic, extendible and configurable with respect to the application domain. The approach focuses on profile and meta-data management, and some MPEG-7/21 based matching components. This way, the user profile allows the additional specification of device characteristics such as the display resolution and the network connectivity like the available bandwidth.

Time and experience in personalization can have a very important role, by granting the moving user a system that anticipates and compensates the time-dependant shifting of user interests [PAS06]. The moving user differs from the desktop user in the way that her or his hand-held device is truly personal. Mobility implies the fact that the user is not bound to a fixed place and to a given time period. Therefore, factors such as time and current activity become increasingly important. A prototype system is implemented to experiment the proposal. It is derived from the *mPERSONA* system and focuses on two new factors: time and experience/activity [PS04]. Thus, only the content description, selection and profile management components were modified in comparison to the *mPERSONA* work. It was shown that the usage of time offers the capability to capture changes in user's interests based on a specific time of the day and adapt its preferences accordingly [PAS06]. Regarding user experience, the article concluded that it provides a way to effectively merge any different instance of the user's profile (e.g., vacation, work, sick), into a dynamic profile. Taking into account the time and current situations, this dynamic profile is able to provide more accurate informations based on the user. The initial

evaluation resulted in a performance improvement between 8% and 173%, depending on the scenario, in comparison to traditional personalization schemes.

2.4.2 Clustering and Context Related Projects

Contextual information is a very important clue for data mining [YW08]. While studying the problem of generating supervision, from the unsupervised training data itself, when the data samples are dependent between each other, the authors refer that the dependency among data is the contextual information. Their work proposes a nested expectation-maximization algorithm that considers the data dependency in a higher level in comparison to traditional k-means clustering.

Wallace and Stamou explain the relation between context and user interest,and also propose a context-aware hierarchical clustering algorithm that is able to mine user interests from multi-relational data sets [WS02]. Given a set of elements (e.g., documents), among which it is defined a variety of dissimilarity measures, the algorithm produces groups of elements that resemble each other, as far as one or more of the given relations are concerned. The algorithm is suitable for applications in which the context is an important factor and the number of clusters is not known before-hand.

Social groups' creation can be employed as a basis for hierarchically structuring user preferences in various domains [VPK+11]. According to the same work: "The successful provision of context aware services entails the attainment of equilibrium between the extent of personalization desired and the user's need for privacy.", and user preferences is an element that plays a significant role in that equilibrium. Some notions and metrics that play a key role in social networking frameworks are described. Among them, there is the notion of semantic distance, semantic proximity, semantic centroid and a proximity measure named Average System Semantic Proximity measures. It is indicated that the work's results can be used in many applications, including personalized media delivery, offering a framework on which next generation multimedia access can be provided.

3

CAPE

This chapter focuses on the proposed personalization environment. It starts by presenting an overview of how CAPE is structured and gives some keyword definitions that will be used frequently along this chapter. In order to use CAPE there are some requirements that need to be fulfilled. An explanation of those requirements is presented, followed by CAPE's data model. After the description of the requirements and the data model, the general architecture is presented, followed by the last section, which explains the main algorithm to generate the resulting personalization.

3.1 Concept and Notions

The main objective of this work is to provide personalization to mobile applications. As mentioned in chapter 1, mobile computing devices have some limitations such as processing power, smaller screens among others, when compared with desktop environments. In order to overcome those limitations, it is necessary to use a technique that is able to profile a user according to his/her behaviour. The solution resides in the adoption of personalization for applications. If a user has access to a personalized application, the presented content should be compatible with what the user really wants or needs, resulting in a better experience.

Machine learning offers several different techniques that are able to provide personalization to applications. Some of the most used techniques are recommendation systems, clustering and classification. There are substantial differences between those techniques. Traditional classification algorithms stand in contrast to clustering algorithms because the later ones are able to decide on their own which distinctions appear to be important

(unsupervised learning algorithms), whereas classification algorithms learn to mimic examples of correct decisions (supervised learning). This learning process implies the need to have labelled training data, i.e., examples of data that have an additional dimension referring to the correct decision for that specific instance of data. Since it is desired to have a generic environment that provides personalization to several applications from different domains, the act of generating labelled data results in a serious limitation, regarding the flexibility concern. Classification algorithms are also different from recommendation algorithms, because they are intended to make a single decision with a very limited set of possible outcomes. On the other hand recommendation algorithms select and rank the best of many possible alternatives and are mostly item-oriented, i.e., they are used in applications that are based on the notion of items, such as stores that sell products or applications that recommend services. This means that recommendation techniques would be useful for some applications and practically useless (or impossible to implement) for other applications that are designed with different purposes.

It is also very important to focus on the user perspective. When users interact with an application they do not behave in a uniform way. Different users have different interests, different knowledge and different capabilities to interact with digital devices. Considering for instance, the ability to interact with mobile devices. A 20 year old user that is used to deal with technology uses the device in a much more efficient way when in comparison with a 70 year old user, that can barely read the smart-phone screen. This means that in this particular situation it is possible to classify those users in two categories: *basic* and *advanced*. Personalization becomes much easier to achieve, if it is possible to assign users to different categories that represent the way they behave.

This work considers this categorization approach as "User Profiling" [ZJL12]. By collecting data related to how a user interacts with an application, it is possible to use that data in a clustering operation with a defined number of clusters, where each one represents a user category. The result of this operation is a set of clusters/categories where each contains a collection of points/users. This knowledge gives information to the application resulting in the ability to personalize it in order to follow each user's image.

Personalization can be represented in a lot of ways, such as modified interface and content, suggestion/recommendation of items and the device's dynamic behaviour. For a given application, the possibilities of personalization can be extremely wide. This open set of different personalization possibilities demands a very generic nature, which makes it extremely difficult to use personalization in a completely automated way. A solution is achieved if each developer decides *a-priori* what parts of the application should be personalized.

Data is crucial in this approach and the application's developer has to choose which data should be used in order to create user profiles. For instance, if an application wants to classify its users by level (e.g., basic, intermediate, advanced) it is important to know how much time users spend using the application. On the other hand, data such as time of access may not be relevant for this kind of personalization. This means that the

application's developer must study which data resources the application is capable of providing, and choose the resources that will in fact be useful.

Although it seems enough, interaction stream data resources by themselves do not provide interesting results due to their simplicity. The ability to build more complex constructions to represent profiles is desirable. For instance, when using the previously presented notion of user level it maybe be not enough to use a single data resource such as the number of logins. An application developer may want to describe the user level as a arithmetic expression composed of variables representing user data, i.e., user level can be defined by a formula like:

$$(w_1 * x + w_2 * y) \tag{3.1}$$

with w, x and y entities being considered as numeric weights and variables. Considering the following values: the first weight to be 0,8; the variable x to be the number of logins; the second weight to be 0,2; the variable y to be the number of clicks on a menu. This would mean that the final value would consist on the number of logins with a weight of 80%, plus the number of clicks on a certain menu with a weight of 20%. This leads to the creation of an abstraction referred in this work as *parameter*, that is a bridge between personalization options and resource data, and enables the developer to use data in a much more flexible way.

As figure 3.1 summarizes, in order to benefit from CAPE, the developer will have to define the desired personalization instances to be used in the application, a set of parameters that will be each used by at least one personalization, and resources that are integrated into at least one parameter formula.

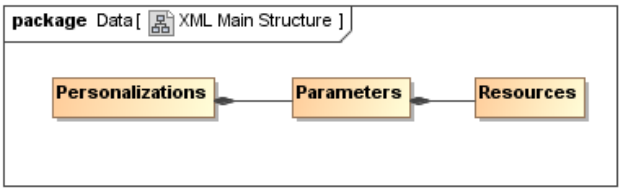


Figure 3.1: Personalization, parameters and resources

3.2 Personalization of an Application

Personalization options, parameters and resources are three main entities of the presented solution, and need to be created and characterized by the application's developer. Due to its expressive capabilities, XML (Extensible Markup Language) technology is adequate for this procedure. In order to use CAPE an XML file must be created to define each resource, parameter and personalization. After the XML file is created, the developer only needs to send it to CAPE's application registration service, so that the application can be registered in the personalization data model. This process enables the application to use the provided personalization API. Figure 3.2 shows the first level of nodes in the XML file.

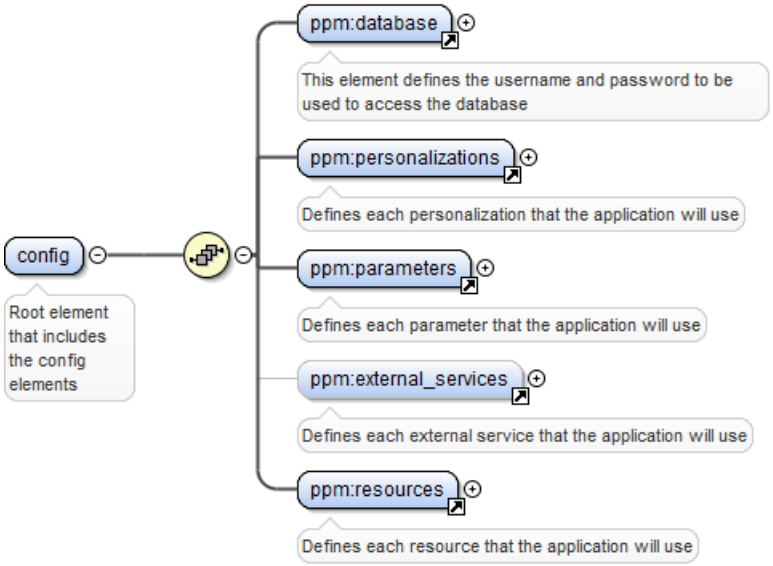


Figure 3.2: Root node of the XML file

3.2.1 User Credentials

Database is the first node of this hierarchy and is related to the definition of the application. As Figure 3.3 shows, it defines the username and password this application will use, when accessing to CAPE's database.

3.2.2 Personalization

The second node, described in Figure 3.4, defines the personalization instances that will be used. Each personalization has a name, a type, the possibility of using context and the listing of all the personalization options to be used. The *type* element is used for situations where a personalization may use different clustering techniques. This work only implements one type of machine learning technique, which means this element is not very useful at the moment, but might be relevant for future work.

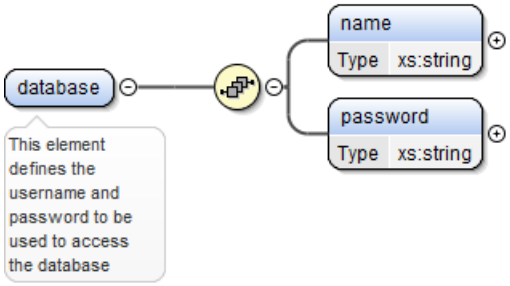


Figure 3.3: Definition of an application in the framework

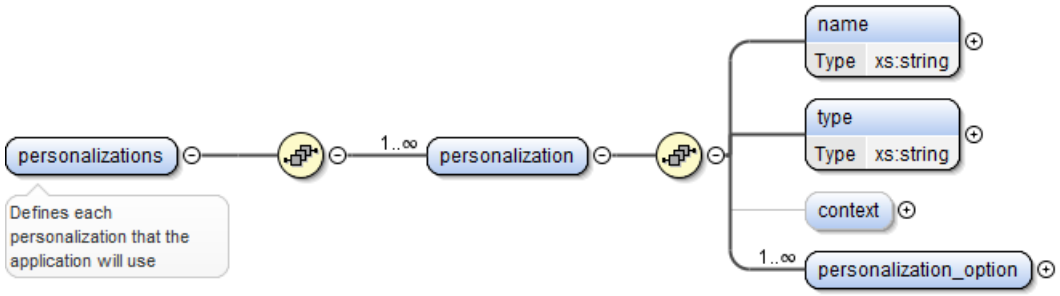


Figure 3.4: Definition of personalization instances to be used

The next element is optional and concerns the usage of context in the application. As referred in section 2.2, context has the potential to refine the used data and make it more accurate for the profile generation. This type of context is referred along this chapter as *context segmentation* because it segments data according to the current captured context. When an application submits a request to update data about a certain user (e.g., number of clicks or number of logins), along with the new value, there is also some information about the sending user's current context (e.g., location or weather conditions). The function that receives this submission updates the respective data values in the database, but also updates those values under the specific context conditions. For instance, user A has 200 logins in total and the application detects that s/he has logged in on a sunny day, the application will send that update to the personalization service along with information that says it is a sunny day. The service updates the database and changes the data value to 201 logins but at the same time updates the number of logins the user did, when he/she was in a context of a sunny day, which obviously needs to be less than 200. This means that context segmentation can be used to refine the personalization results because in certain situations, the current context has a direct influence on the way a user thinks and interacts.

Context is described in Figure 3.5, which shows that this element is composed of another element named *total context weight* and a set of *context resources* that have a name and a *context weight*. The name element specifies what kind of context will be used (so far CAPE supports two types: current temperature and the current hour of access) and the context weight identifies how much, does a context element weight, in comparison

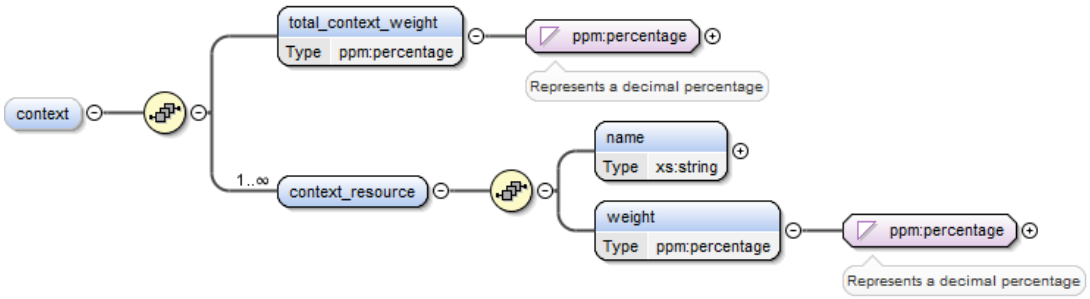


Figure 3.5: Definition of the usage of context in a personalization

to other context elements. The sum of the weights of all context resources in a given personalization must always be 1, i.e., each weight indicates a percentage. The usage of context at this stage follows an in-model approach because the data itself is modified before generating the user profiles. It works in the following way:

The operation starts with a personalization request that is sent with information concerning the application's current context (e.g., time and location). CAPE's database stores resource information that is conditioned by context, and that data is retrieved in order to be weighted with assigned weights for each context. The result of the weighting operation is multiplied by *total context weight* value. User data not restricted by context is retrieved from the database and multiplied by *(1-total context weight)*. The non-contextual result is arithmetically added to the contextual result, forming the final value that will be used for clustering purposes. The following formula defines it:

$$\begin{gathered}[(1 - TotalContextWeight) * (NonContextData)] \\ + \\ [(TotalContextWeight) * (W_{c1} * C_1Data + W_{c2} * C_2Data)]\end{gathered} \tag{3.2}$$

Example: Supposing that a user was using an application that made a request to CAPE. That request happened at 12h:34 with 19ºC outside. That application developer, while configuring the desired personalization, decided to use a profile named *user level* that divides users as *basic user* and *advanced user*. The profile data is defined by the amount of time a user has spent logged in. The developer also decides that this operation must very accurate and concludes that the time of the day and the current outdoors temperature affects the amount of times a user logs into the application. According to the application's nature, the developer decides that these two context variables should have a total influence of 25% on the value that characterizes a user, i.e., the variable *total context weight* has a value of 0,25. The developer also thinks that the current hour has a bigger influence on how much a user logs into the application, in comparison to the current temperature, which is why the hours are assigned a weight of 0,6 and the temperature a weight of 0,4. Supposing a user that has spent a total of 60 minutes logged-in. Of those 60 minutes, 40 minutes were spent between 09:00-17:00 and only 20 minutes

between 17:00-09:00. Regarding temperature, 55 minutes were spent with temperatures smaller than 21ºC and only 5 minutes were spent with temperatures equal or bigger than 21ºC.

Description	Value
Personalization:	User Level, defined by total logged-in time
Total log-in time:	60 minutes
Log-in time between 09:00-17:00:	40 minutes
Log-in time when below 21ºC:	55 minutes
Log-in time when above 21ºC:	5 minutes
Total Context Weight:	0,25
Hours weight:	0,6
Temperature weight:	0,4
Current time:	12h:34
Current temperature:	19ºC

Table 3.1: Summary

Since it is 19ºC outside, CAPE will use data that was collected for temperatures below 21ºC, which means that the 55 minutes value will be used. The time of request marks 12h:34 which means that the 40 minutes value will be used. Replacing all the appropriate values in the previous formula, the result would be:

$$[(0,75) * (60)] + [(0,25) * (0,4 * 55 + 0,6 * 40)] = 57,25 \tag{3.3}$$

The returning value of 57,25 would be be used in the clustering operation, for the current user.

Following the optional definition of context comes the personalization options that will be used to define a personalization. Figure 3.6 shows that each personalization option is defined by a list of parameter options and possibly external service options. For example, a developer may want to use a personalization in which the options depend on the parameter *user level* (basic, intermediate or advanced) and on the external service *weather* (measures the current meteorology condition). These personalization options are the information pool, from which one will be selected and sent to the user as an answer to a personalization request.

3.2.3 Parameters

The third node is related to the definition of the parameters and it is presented in Figure 3.7. A parameter is characterized by a name, that is used to uniquely identify the parameter and is also used in the definition of personalization options. This node contains a set of ordered options that identify the clusters that will be used in the clustering process. It is important to point out that the order in which the clusters are defined is crucial to get the correct results. When defining a parameter the developer must be aware that the first parameter options should be semantically equivalent to lower values, while the last

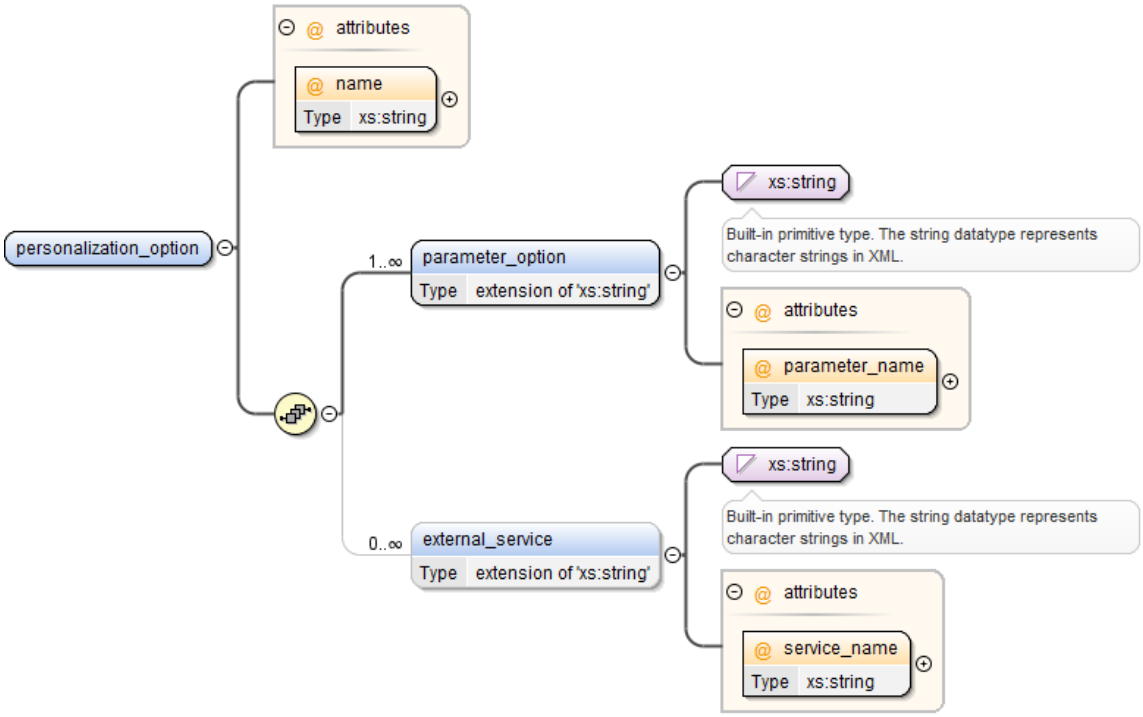


Figure 3.6: Definition of a personalization option

parameter options should be semantically equivalent to higher values. For instance, if a parameter is used to profile users in 3 levels (basic, intermediate and advanced) and the used data consists on the sum of time each user spent logged-in, then a user with a small value will be considered *basic*, and another user with a greater value will be considered *advanced*. This happens because both of them have dedicated different amounts of time to the application. On the other hand, if the *advanced* option was defined before the *basic* option, users with a small value would be considered *advanced*, and users with a larger value would be considered *basic*, which is not correct.

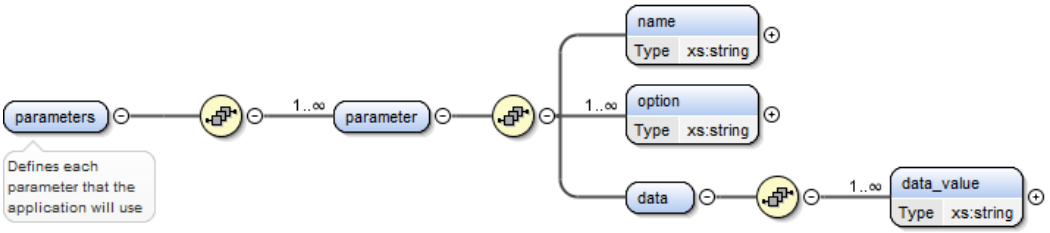


Figure 3.7: Definition of all the used parameters with the formulas and parameter options

The developer may create one or more different arithmetic formulas in the parameter definition. Those formulas currently support integer and double type values, basic arithmetic operations (sum, subtraction, multiplication and division) and the usage of parenthesis to express arithmetic priorities between different expressions. It is important to point out that CAPE is prepared to deal with serious mathematical problems such as

division by zero, by replacing the value with 0.

In order to perform the clustering operation, the algorithm needs input, normally in the form of a vector. CAPE uses the same approach. Each index of that vector represents a user. Each parameter formula represents a different dimension in the vector of data that will be used in the clustering process, i.e., instead of having a vector of users with only one value used in the clustering process, a matrix of users is used, where each user may have more than one value, which concern to other different formulas. For example, it is desired to execute a clustering process to check whether a user likes football and baseball, the first sport can occupy the first vector dimension and the second sport can occupy the second vector dimension. This can be interesting to verify if a user likes ball-based sports. Table 3.2 shows an example of this situation. Observing the values it is possible to conclude that user 0 does like ball-related sports, while user 4 does not like at all, ball-related sports.

User	Data Dimension 1 - Football	Data Dimension 2 - Baseball
0	89	94
1	21	74
2	98	3
3	89	17
4	3	1

Table 3.2: Input using two data dimensions

3.2.4 External Services

Figure 3.8 outlines the fourth node that is used to define which external services will be used in the application. As with the parameters, a name element is used to identify the external service when defining the personalization options. The next element is used to classify the type of external service, selected from a collection of limited types provided by CAPE. CAPE can use multiple external services but so far, the only offered service is related to meteorology using the web-service *World Weather Online*, outlined in section 2.3.3. The last element aims to forearm situations when the external service is unavailable. A default value is defined depending on the type of external service, i.e., the element is coincident with one of the available external service options. For instance, if the weather service was unavailable and a personalization depended on it, the later would use the defined default value (e.g., "sun").

3.2.5 Resources

The last node in the XML file marks the used application's resources. Diagram 3.9 shows that CAPE considers three types of resources: *Interaction stream*, *context* and *preferences*.

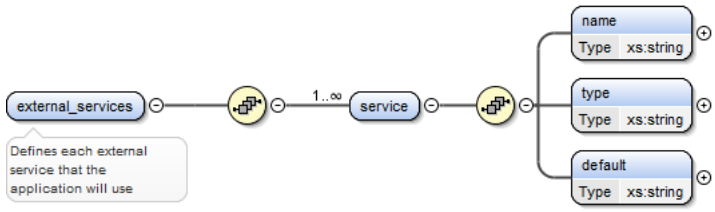


Figure 3.8: Definition of all the used external services

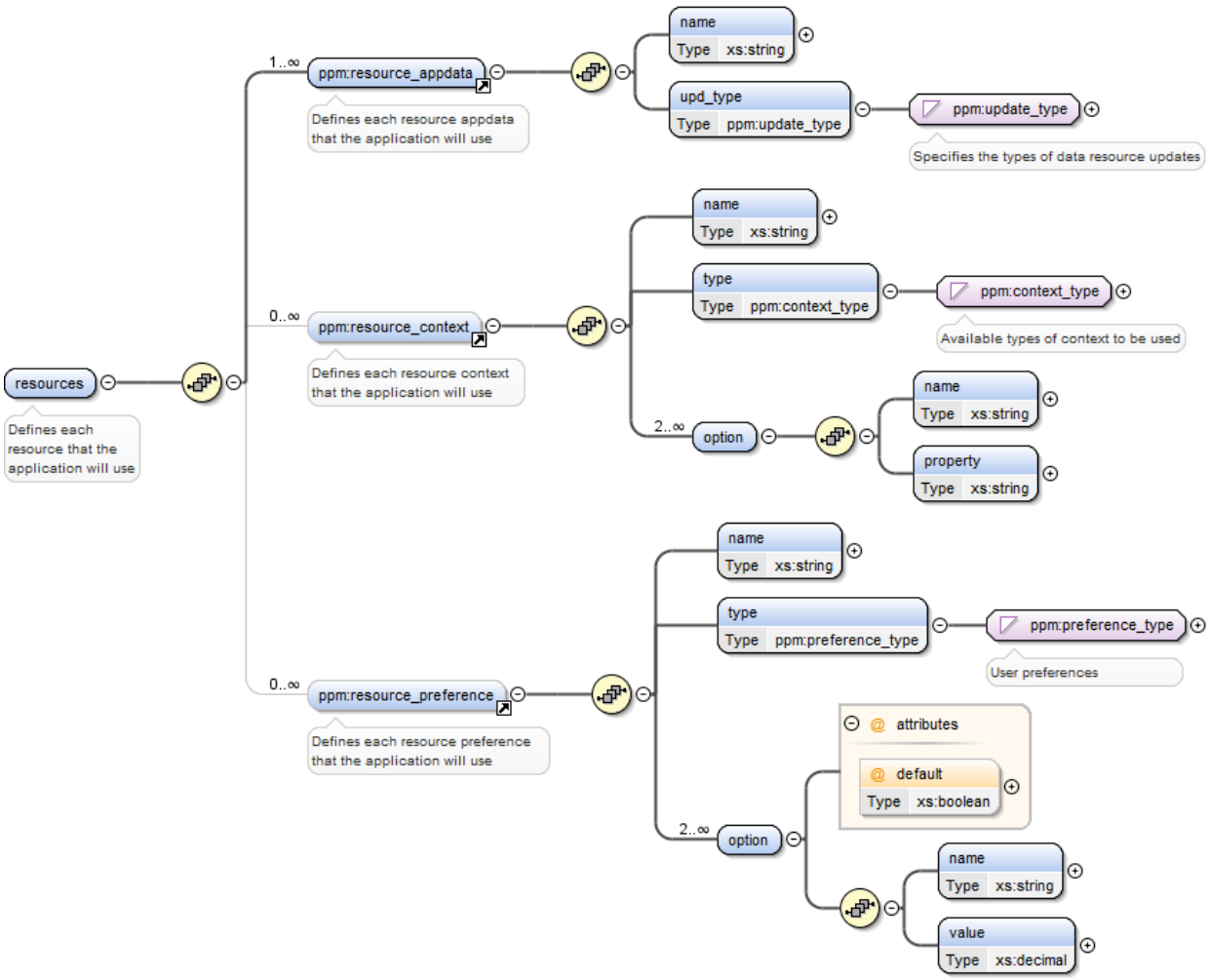


Figure 3.9: Definition of all the used resources

3.2.5.1 Interaction Stream

The first element, referred as *resource appdata* addresses interaction stream data, i.e., data obtained from the normal interaction of each user with the application. Number of logins and time logged-in are examples of this type of resource. As the example shows, the number of logins is updated incrementally and the time logged in is a sum of time spent while using the application. This means there can be different types of data updates. This is the reason why the developer must specify, from a limited list, what kind of update type will be used for a given resource. The CAPE prototype is prepared for two update types: incrementation by one unit and addition of values.

3.2.5.2 Context

The second element addresses data concerning the context of the application's usage. Each context resource has a name that identifies it and also a type to describe it. Following that, there is a set of options that depend on the chosen resource context type. Those options are used to describe how many context options will be used in any personalization process that uses context segmentation. The amount of options depends entirely on the level of detail the developer wants to use. Each context option has a name and a property that is used to define the circumstances under which, that context option should be applied. For example, if there is a context resource of the type *hour interval*, the developer will choose the amount of options, appropriate for the application. Each context option should have a property that expresses the hour interval like 08:00-14:00.

3.2.5.3 Preferences

Nowadays, most applications save information about the user, which is not supposed to change frequently. That kind of information is usually named as *User Preferences*, representing the third resource element. This resource type is supposed to be less dynamic in comparison to context data. It has a name element to identify the preference, a type of preference like open-text or multiple-choice, and a set of options that may be used to describe multiple choice preferences. For the situation of a multiple-choice preference, the number of options would be equal to the number of preference choices. Each option also has a name that identifies it, a numeric value and a boolean attribute that indicates if the option is to be used by default (in case the user does not specify a preference). The value element can be used arithmetically (sum, subtraction, multiplication and division) in parameter formulas just like any other interaction stream resource, which is particularly interesting to refine values. Considering the formula 3.4, that defines a parameter which represents the level of a user between *basic* and *advanced*, it is possible to observe the presence of two variables. The first variable is the most important one, related to the amount of time a user has spent logged into the application. It is considered as an interaction stream variables. The second variable concerns a user preference and has a more static nature because it does not change as often as an interaction stream variable.

Assuming that variable *pref1* is a multiple-choice question, from which the answer influences the level of a user. If for example, the user chooses an option (choice) with value 1,1, this same value can be used arithmetically with the rest of the formula (variable *timeLoggedIn*. In this example the respective parameter formula would suffer a numeric increase of 10%, due to the multiplication by 1,1, which implies a value refinement.

$$ParameterDataFormula = timeLoggedIn * pref1 \tag{3.4}$$

3.3 Personalization Data model

The objective of this section is to define and lay out CAPE's data model. Given its complexity, the data model is divided in two different parts, that interconnect between each other in order to simplify the reading. The first part concerns data related to applications and it consists on the data extracted from the XML files that are used to register applications. The second part concerns data related to users and their interaction with the respective applications. The complete diagram can be seen in appendix B.

3.3.1 Application

The application data model, presented in this sub-section focuses on storing the configuration details of every registered application. This configuration is obtained through the XML file sent by every application's developer and can be broadly divided into: *Personalizations*, *Parameters* and *Resources*, following the XML Schema structure presented in chapter 3.2.

The *Application* entity figures in the centre of the data model and works as a "hub" among other entities. It stores the basic information related to each application registered in the database. The fields *application name* and *password* are used as authentication credentials to interact with the database.

3.3.1.1 Personalization and Parameters

Figure 3.10 presents the data model structure concerning personalization (green area) and parameters (red area). Starting with the green area, the diagram shows that linked to the previous entity there is the *Personalization* entity, which contains data about each personalization used in an application. Each personalization has a *name* attribute used as identification inside the application. There is also an attribute named *context segmentation* of the type boolean, which is used to differentiate personalization instances that use context segmentation from those that do not use it.

As previously referred in section 3.1, a personalization needs options to represent each possible user profile, which justifies the existence of the entity *Personalization option*. The third attribute named *clustering profile id* is used to link this entity to the entity *Clustering profile*, which is a simple optimization for situations where different personalization

options share the same combination of parameters in use. This optimization prevents CAPE from doing unnecessary calculations on data that was already used previously.

Personalization options are built, based on combinations of parameter options that are obtained using the clustering operation. Already in the red area, the entity *Personalization parameter option* is used as an intermediate entity to store the combinations of parameter options provided by the entity *Parameter option*. The entity *Parameter option* stores all parameter options that were defined along with their respective parameter. A parameter option can also be considered as a cluster of data. Each parameter option is identified by a combination of its declared name and the respective parameter identifier, which means that different parameters can share parameter options with the same name. As referred in the previous section, a parameter can have one or more arithmetic formulas that will be used for the clustering operation, and those formulas are stored in the entity *Parameter data*, in the form of a string. Each formula must use, at least, one variable referring to interaction stream resources or user preferences[1]. In order to calculate the numeric value from each parameter formula, the algorithm parses the string, identifies the existing variables (resources), replaces them with their respective values and obtains the final result that will be used in the clustering process.

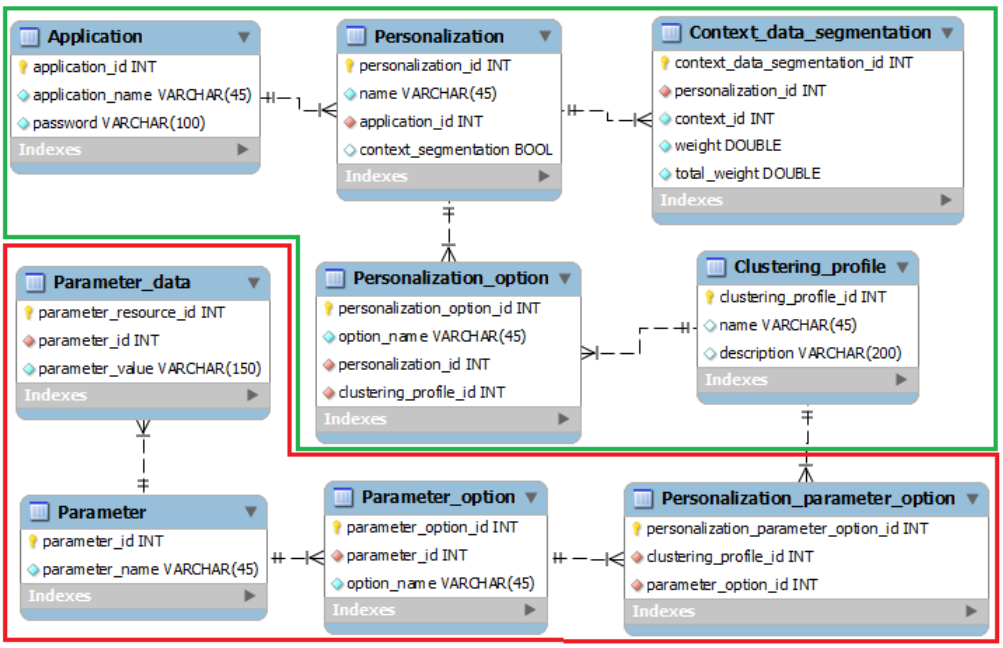


Figure 3.10: Personalization instances and parameters

[1]This means that there is a relation between parameters and resources that is not present in the data model because it is not trivial to represent mathematical formulas in this kind of modelling, and would turn the data model even more complex.

3.3.1.2 Resources

Resource is the entity that stores the basic data elements that are obtained from each single application. Each element can assume one of three types: Interaction Stream (red area), Preference (green area) and Context (purple area). These elements are shown in diagram 3.11.

Resource interaction stream entity is used to store normal interaction stream data such as number of logins or number of clicks in a certain menu. Interaction stream resources can greatly differ from each other, which indicates the need to assign to each resource a type of data update. The entity *Resource updtype* provides types of data updates that can be assigned to interaction stream resources.

Resource preference is an entity that represents another type of resource. It is meant to store the preferences a user expresses in each application s/he uses. Preferences are not universal and can also assume different types, such as multiple choice with a variable amount of options or even just plain text. The *Preference type* and *Preference option* entities are used to express these different types of preferences. While the first entity is simply a collection of preference types supported by CAPE, the second entity provides different possibilities of preference options, which can be used for multiple choice purposes. In order to store this kind of information, *Preference option* entity uses an attribute named *value*, to store numeric value, and also uses an attribute named *property* that is used to represent any kind of additional information, depending on the type of preference.

The third resource type is named *Resource context*, as an entity that represents the several possibilities of context segmentation applied to a personalization. Each context resource needs to be assigned a context type, because there are several types of context, such as temperature or hour of access. That collection of context types is stored in the element *Context type*. A developer has the possibility to choose different options for the context choice. For instance, if taken into account the usage of temperature as a context resource, the developer may use 2 or more options to define the temperature intervals. This means that the number of options is directly related to the level of context precision the developer is looking for. Those options are stored in the element *Context option*. In this element, the attribute *property* is used to define a criteria for the context, i.e., if using a context type such as hour interval, the property attribute would be used to express that interval. It was referred previously that CAPE is currently using two possible types of context: hour interval and temperature. The first one uses a property syntax that expresses the interval with the minimum time in the 24-hour format, separated by a dash from the maximum time, also in the 24-hour format (e.g., 09:00-17:00). Regarding temperature, the property is expressed using the minor and major arithmetic symbols (e.g., 17^{o}<23^{o} or >21^{o}).

Resource context is not a user-related resource, but is related to each personalization. When configuring the personalization instances to be used in the application, each developer has the chance to choose if a certain personalization will benefit or not from context.

If context segmentation is used, the entity *Context data segmentation* will store the relation and the respective data. Figure 3.10 shows that, besides storing the personalization id and the context id, this entity also has a numeric attribute named *weight* associated with each context option, and also another numeric attribute that represents the total weight, those context options will have in the final calculations. The previous section explains how these values are used. This type of context changes the results in-model (technique referred in section 2.2.3), i.e., the numeric values that feed the clustering process, are refined according to the current context.

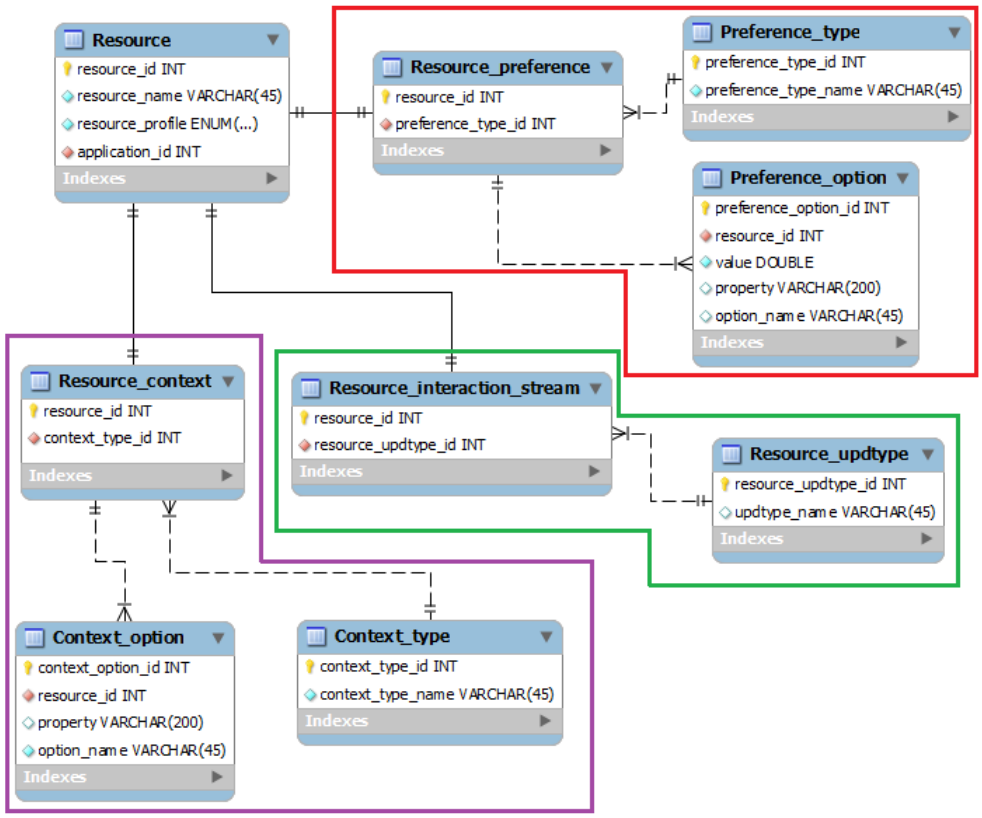


Figure 3.11: Resources

3.3.1.3 External Services

Figure 3.12 shows that CAPE offers another type of context intervention, named External Services and represented by the green area. Instead of modifying data that is used in the clustering process, this approach is used to filter the final results post-clustering. The entity *Personalization option external service* stores the different combinations of personalization options and external service options. When a user makes a personalization request, the algorithm will check which personalization option was chosen after the clustering process was executed, and with the current request context will submit to the user the right combination of the personalization option and the external service option. For example, supposing that an application makes a request and the current context indicates that it is raining in the detected location. After the clustering process, the requesting user is flagged with a *basic* profile, and if the developer specified that this personalization is defined to use an external service, then the result will have a *basic* profile with the flag that it is raining. In practice the application will receive a different profile as an answer to the personalization request.

External services also need to have options specified, depending on the type of context that is being used. According to section 2.2.3, this approach can be considered as post-filtering context. Each type of external service is stored in the entity *External service type*. Depending on the type of context, there may be the need to have several levels to differentiate them. This differentiation is useful for situations where a developer wants to use more or less detailed options. For instance, a developer might want to use a meteorology external service, but in a very basic level (e.g., sun, rain, clouds) and another developer might want to use the meteorology external service in a more detailed level (e.g., light rain, heavy rain or thunderstorm), which implies the need to create an entity called *External service level* which stores different levels of context. Each of these levels also need to store the several different external service options they offer, which makes the entity *External service option* quite appropriate for this objective.

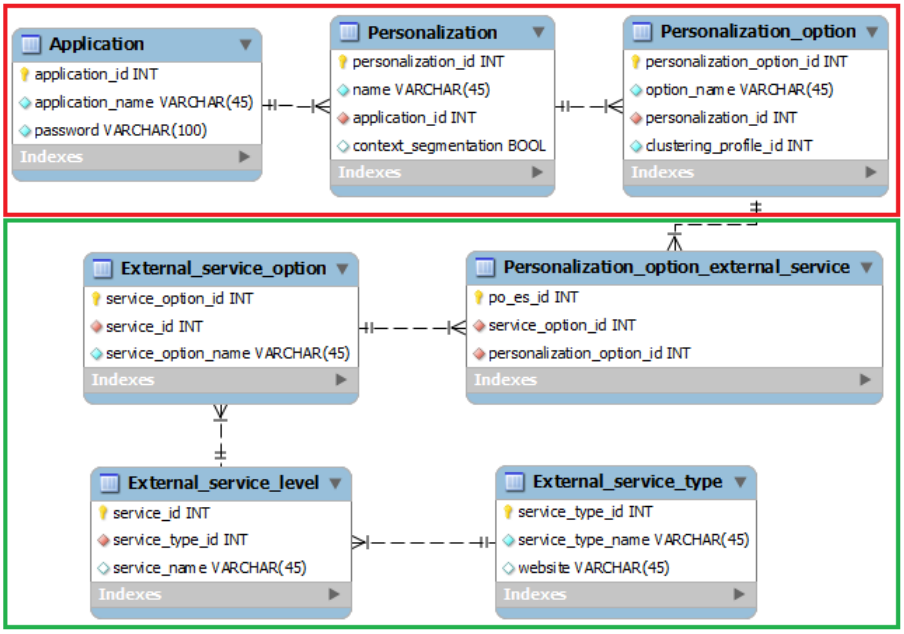


Figure 3.12: External Services

3.3.2 Users/Interaction

The users/interaction data model, shown in Figure 3.13, is a sub-model of the complete data model, that focuses on storing data associated with each user and application. Along with the application data model, the *Application* is the central entity. Linked to it, there is the entity *Application user* that stores the applications each user is registered, which implies that a user may be registered in more than one application. As the diagram shows, the email used by a user identifies him/her, but there may be more than one username, depending on the application in which the user is registered.

The most important part of this data model concerns to each user's application usage data, i.e., the data obtained from the user's interaction with the application and also his/her personal preferences within the application. *User resource data* is an entity that saves every user's interaction stream data, which is why it is connected to the entities *User* and *Resource*.

While this would be enough for applications that use personalization without any kind of context segmentation, there are some applications that indeed benefit from the usage of this technique. As referred previously in section 3.1, context segmentation stores and updates data values, taking into account the current context of access of request. The red area represents the entities that implement this solution. The entity *User data context* is used to store data conditioned by context. Because it is linked to *User resource data* and the *Context option*, it is possible to combine users with context options and the respective segmented data that is needed to execute the clustering algorithm. Entity *User profile* stores the result of each personalization request (it does not take into account post-clustering context) along with the date and time of the request. This entity is linked to the entity *User* and the entity *Clustering profile* in order to access the existing users and personalization options of each personalization.

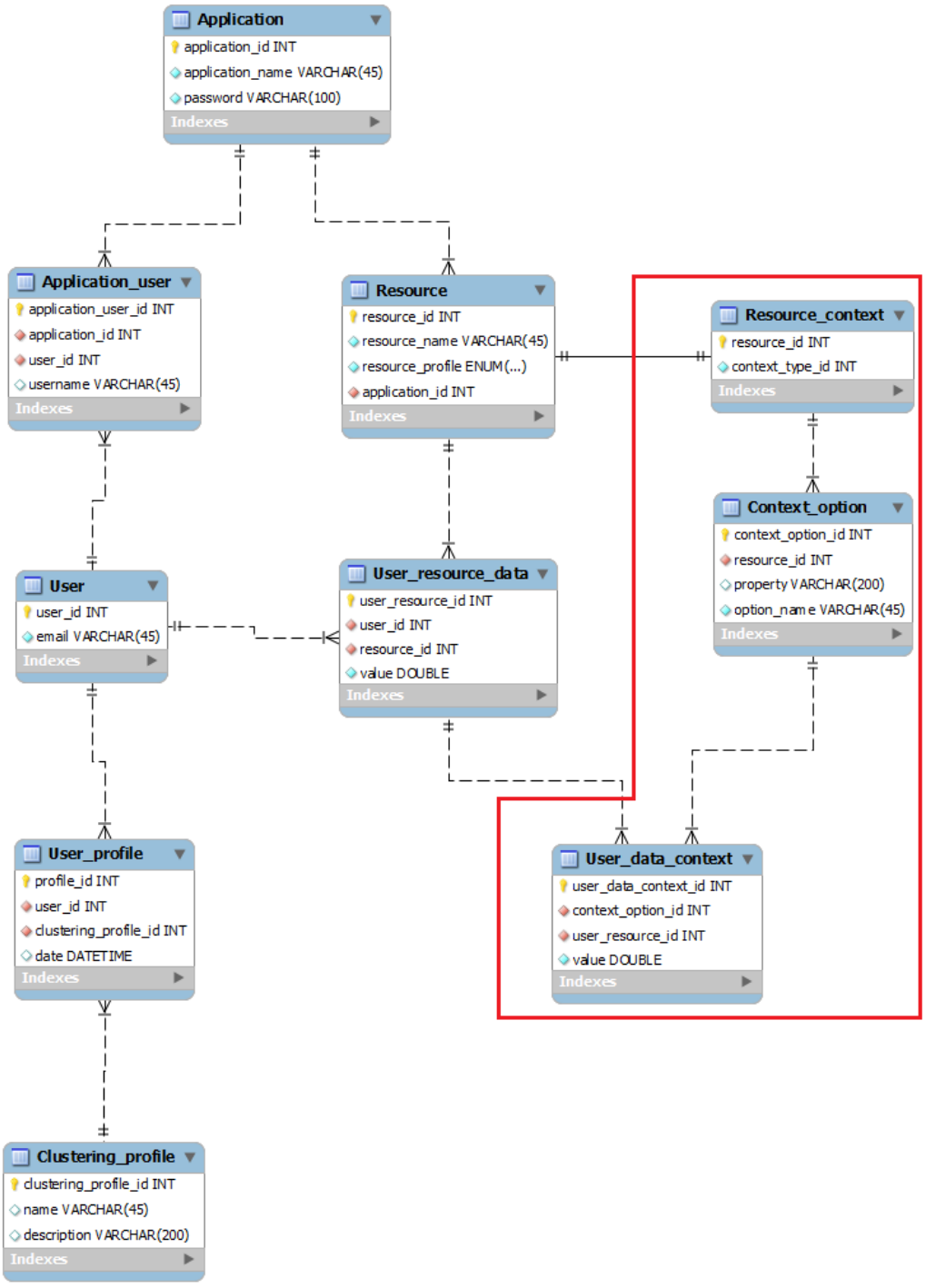


Figure 3.13: Data model of the users/interaction data

3.4 Architecture

CAPE follows a client/server model architecture. A client/server is often a generic umbrella term for any application architecture that divides processing among two or more processes, often on two or more machines [Ree00].

For the presented case-study purposes (see chapter 4), CAPE was developed in order to be a web-service that works upon HTTP (Hyper Text Transfer Protocol) requests.

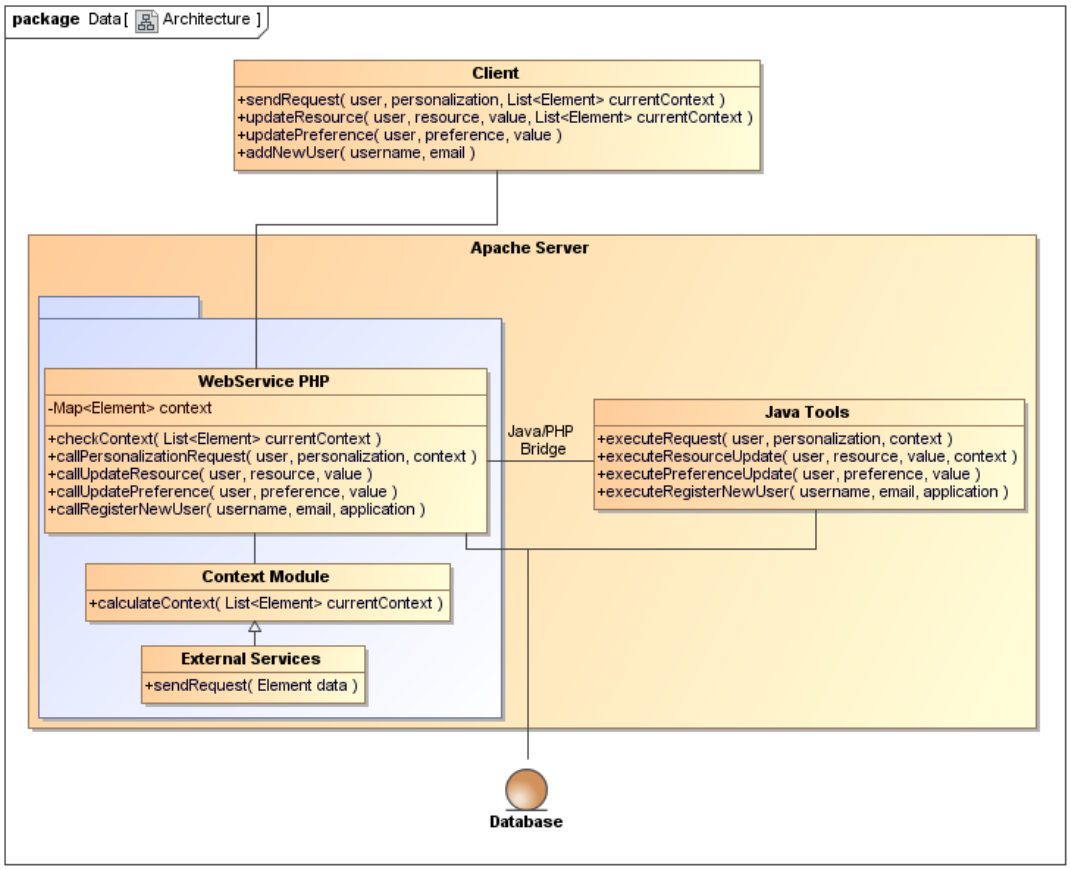


Figure 3.14: CAPE architecture

Figure 3.14 shows the current architecture diagram. It can be considered as a 3-tier client/server architecture, although the first two tiers can be considered as interconnected between each other because some of the logic is shared. The first tier, also considered as CAPE's interface receives the HTTP request from a client and forwards the request to the respective Java tools that are considered in the second tier. This forwarding procedure is shown in the diagram with the subtitle "Java/PHP Bridge" because the implemented case-study (see chapter 4) used PHP as the web-service script language. In order to create communication between PHP and Java, CAPE is using a module called Java/PHP Bridge[2] that is an implementation of a streaming, XML-based network protocol, which can be used to connect a native script engine (e.g., PHP, Scheme or Python), with a Java

[2]http://php-java-bridge.sourceforge.net/

virtual machine. Finally, the third tier is the database, which stores the data concerning each application and each user, to execute the operations.

The first tier is used to redirect personalization requests and other operations concerning personalization, to the second tier of the architecture, which implements the logic part. Despite this main function, the first tier also has other functions such as obtaining context information. Section 3.2 indicates that a personalization may use context approaches (Context Segmentation or Context External Services) to refine the user profiling results. Those context approaches may need information that is only obtainable outside of the system. For example, an application may need to access a weather forecast in order to improve personalization. Given GPS coordinates as input, a context module associated with the first tier of architecture would use that data to handle this procedure and provide the right context information to the machine learning algorithm. The *Context Module* may use an external service to obtain more information, but there are types of context that do not need an external service. For instance, when using hour intervals as context, the context module only needs the hour of request, making a match between that hour and the previously defined hour intervals.

3.5 CAPE's algorithms

As it can be inferred from figure 3.14, a normal interaction with CAPE consists on an application sending a request, which is received by the server's scripting language files that will do some data preprocessing and redirect the request to the logic implementation. The logic part will calculate the desired outcome, using the database to obtain necessary data, and send the result backwards to the application client. This section describes the structure and behaviour of the logic part.

3.5.1 Personalization Request

Figure 3.15 shows the main algorithm that is used to calculate a user profile. Upon receiving a request for a certain personalization, the algorithm will have access to the following parameters:

- Database user credentials;
- User id;
- Personalization id;
- Associative array that stores the user's current context of access to the application.

Initially, using the provided user credentials, the algorithm logs-in into the database to have access to the user's and application data. After this procedure, the algorithm will retrieve from the database, information related to the requested personalization, followed by the respective parameters that the personalization uses. The diagram is divided in two parts: Personalization that uses context segmentation, and personalization that does not use context segmentation. Although both approaches are similar, there are some important differences to point out.

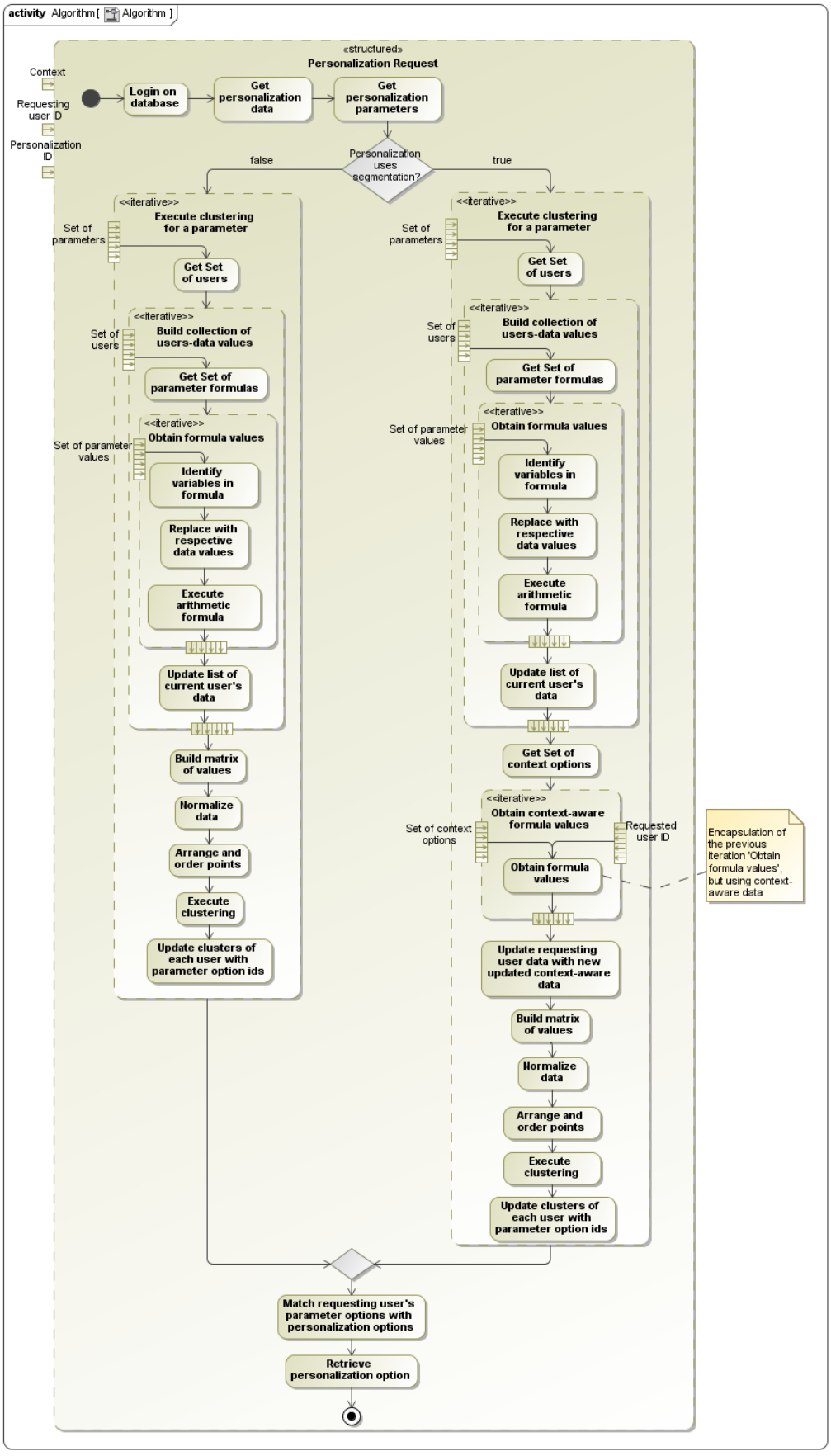


Figure 3.15: The main algorithm that calculates user profiles

3.5.1.1 Without Context Segmentation

The algorithm starts with an iteration of every parameter used by the personalization instance. For each parameter in the iteration, the set of users registered in the application will be iterated, and for each of those users there will be an iteration in the formulas of the current parameter, because a parameter may have more than one formula, depending on how the application developer configured it. At this point, the algorithm has the current parameter id, user id and parameter formula in iteration. This data allows the possibility to identify the variables in each parameter formula, replace them with the corresponding data values and execute the arithmetic formula. The resulting value identifies the type of user and is saved in a local data structure. When all users and parameter formulas are iterated, the mentioned data structure will be holding each user and the respective values for each parameter formula. This data structure will be iterated and converted into a matrix where each line represents a different user and each column a different parameter formula. In this situation, the values in the matrix are not normalized. For instance, one dimension may assume values between 0 and 1 while a second dimension may assume values between 1000 and 2000. If distance measures such as the Euclidean distance were in use, the outcome of the clustering process would be much more sensitive to the second dimension due to its wide range and higher values, i.e., the first dimension would be overlooked. While there are many types of data normalization techniques available, CAPE only uses the *Min Max Normalization*, because it provides the means to solve the normalization problem while standing for its simplicity. Min Max Normalization transforms a value A into B which fits into the range [C,D] and is given by the following formula:

$$B = (\frac{(A - minimumvalueofA)}{(maximumvalueofA - minimumvaluesofB)}) * (D - C) + C \tag{3.5}$$

After the data normalization procedure, the order in which data points are submitted to the clustering algorithm needs to change, so that the order in which the parameter options are established may be respected. For example, using a parameter called *user level* that has three different possible clusters named *basic*, *intermediate* and *advanced*, the order in which these clusters are established needs to be respected because the clustering algorithm has no semantic knowledge about those options. While a *basic* cluster is linked to a smaller value, an *advanced* cluster is linked to a bigger value, which means that cluster 1 should be assigned to *basic* users, cluster 2 assigned to *intermediate* users and cluster 3 assigned to *advanced* users. The used implementation of K-means algorithm is the one implemented by Weka framework. The implementation chooses the initial clusters randomly from the entire list of points, which is not helpful at all for the desired purposes. Given this problem, the K-means algorithm that is being used in CAPE, was slightly modified in order to choose the first *n* points that appear in the submitted matrix of data, with *n* being the number of used clusters in the clustering execution. The arrangement of points consists of the ascending ordering of all the points, dividing the array in *n* parts

and the first point in each part will be transferred to the respective position in the beginning of the array. Figure 3.16 shows an example of this situation. Using the array in point A and using three clusters, the arrangement would order all the points, resulting in the array in point B and the n-division would select points 1, 7 and 17, as seen in point C that would be transferred to the beginning of the array. Finally, the array to be submitted to the clustering algorithm would be the one in point D, with cluster 0 starting with point 1, cluster 1 starting with point 7 and cluster 3 starting with point 17.

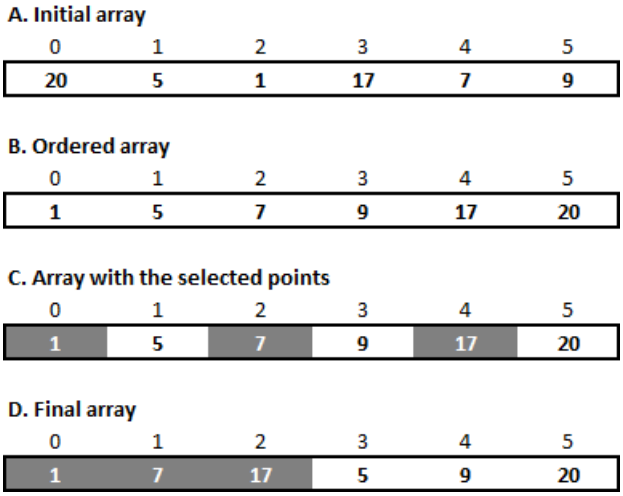


Figure 3.16: Array ordering

After executing the clustering algorithm, the results show that points are assigned to clusters numbered from 0 to n-1, which means a conversion will be made to the respective parameter options. Having a data structure composed of users and their respective parameter options, a matching of which personalization option corresponds to the combination of parameter options is made. For instance, personalization option X can only be given to users who have parameter options $X1$ and $X2$, depending on how the application developer configured the application.

When the personalization option is obtained, the CAPE will send a response to the user with the desired profile for the requested personalization.

3.5.1.2 With Context Segmentation

For personalization instances that benefit from context segmentation, the algorithm is quite similar to what was presented previously but it is marked by some differences. The data structure of users and respective parameter formula values is still populated in the same way as previously, but instead of sending it immediately to the clustering operation, the algorithm makes some modifications to the user data. For each context option that the requesting user was assigned (being alone, at work or during night-time are examples of three different context options) instead of retrieving from the database the normal user's interaction stream, the data that is retrieved is conditioned by the user's current context.

For instance, supposing that the current parameter formula uses the number of logins as a variable and the user has registered 200 logins in total. Those 200 logins were collected context independently, but the database also stores the number of logins conditioned by context and instead of using those 200 logins, the conditioned value (less or equal than 200) will be used in the operation. Now that the data structure is modified with the context-aware values, the clustering algorithm can be used to deliver more accurate results.

3.5.2 Other Operations

The personalization request method needs an environment with users and their respective data, in order to be used with success by an application. CAPE's API offers three more methods: User Creation, Resource Data Update and User Preferences Update.

3.5.2.1 User Creation

In order to provide personalization to applications, CAPE needs users and their respective data. The method described in figure 3.17 shows the activity flow of this algorithm. The program verifies if the provided email already exists in the database, and if that is not the case a new user will be registered, both in the table of users and the table that combines users and applications (remembering that a user may use more than one application). On the other hand, if the user is already registered in the database, only the table that combines users and applications needs to be changed, i.e., a user gets registered in the corresponding application. After these steps, the user's resource values need to be initialized to 0 as well as the user's preferences need to be initialized to their default values.

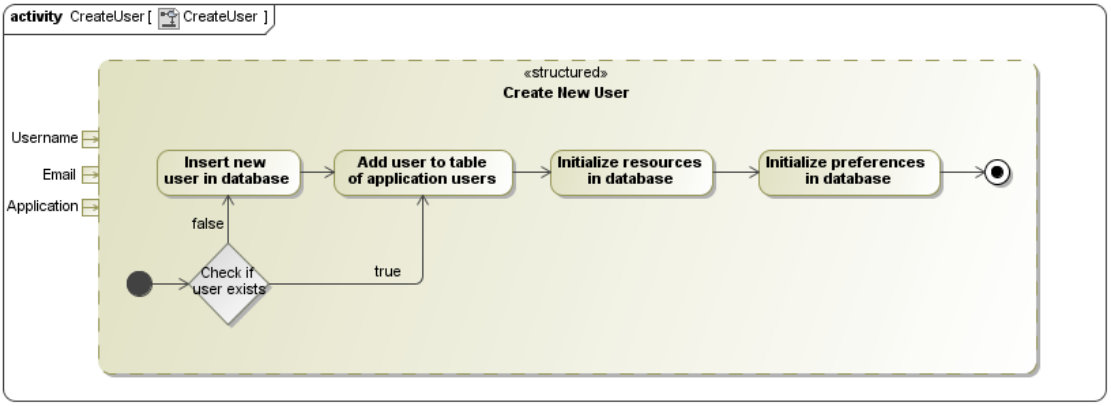


Figure 3.17: The algorithm to create new users

3.5.2.2 Resource Data Update

The method to update the user's resource values starts with a validation of the type of resource update (e.g., incremental, addition). After that verification, the current resource value is retrieved from the database and updated with the new updated value, and finally it is stored in the database. This would be enough if context segmentation was not an issue, but since there is a possibility that an application is using context segmentation in one or more personalization instances, it is also required to update the resource values under the context in which the user is updating them. For instance, if the user logs-in into the application at the workplace, a request to update the number of logins is sent, and besides updating the global number of logins the user has already made in the past, it also updates the number of logins that were made when it was detected that the user was at the workplace. Using the user's current context, the algorithm updates the respective context-aware resource values with the new values and stores it in the database.

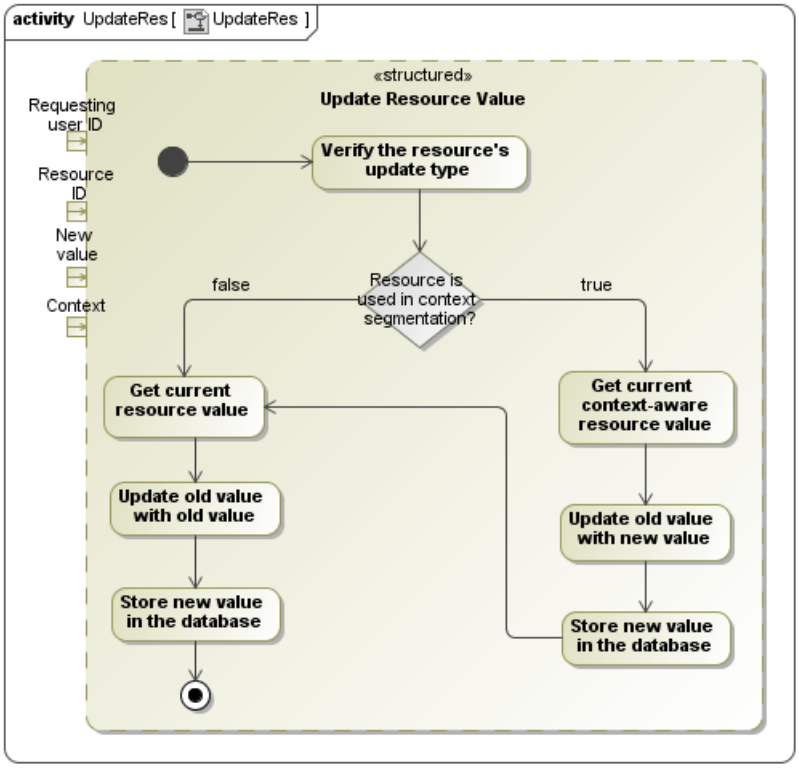


Figure 3.18: The algorithm that updates resource values

3.5.2.3 User Preferences Update

Previously, it was referred that CAPE offers the capability to use the concept of user preferences. Those preferences will probably change with time and CAPE needs to be able to handle those updates. The algorithm is extremely simple, as it consists on verifying the preference type (e.g., multiple choice, plain text), updating the old value with the new

one, following the preference type directive, and storing the new updated value into the database.

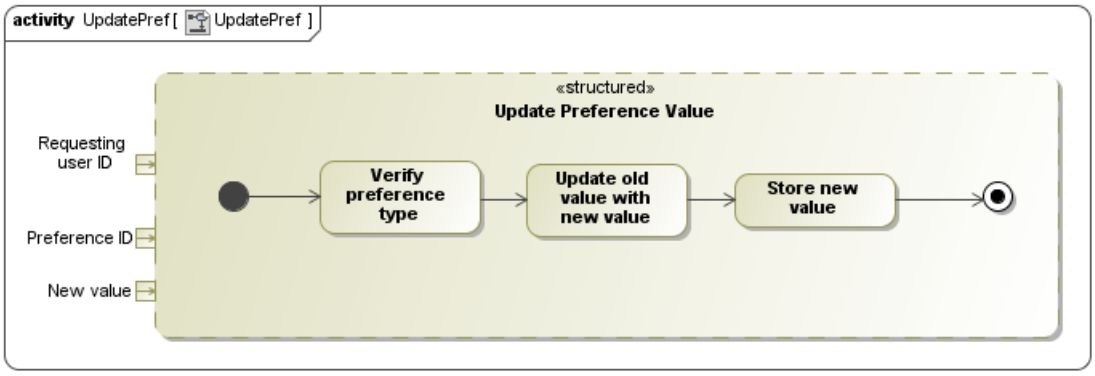


Figure 3.19: The algorithm that updates the user's preferences

3.5.3 Scripting Language Files

These files are used as the interface that receives the requests from CAPE's applications. Besides redirecting the requests to the previous algorithms, they also build a context dictionary that is required by those same algorithms. A mobile application when using CAPE's API may also send potential context data, such as current time and/or GPS (Global Positioning System) coordinates. The scripting files use that data to build context information by identifying the user's current context, in conformity to what was defined by the application. For instance, if an application is using a type of personalization that requires information concerning the current temperature, then the submitted GPS coordinates will be sent to an external weather station in order to obtain the local weather. When the context becomes known, the algorithm matches the results with the context options, that were defined upon the application registration within CAPE. The result of this matching is a list of context option ids that will be used by the logic-tier algorithms, to identify the user's current context.

4

Case Study - LEY

This chapter contains a case study applied to the mobile application LEY, using CAPE. The first section presents an introduction to the mobile game, the second section the personalization solutions that were chosen and the final section a discussion concerning what was achieved with the study.

4.1 Introducing LEY

LEY, from Less energy Empowers You is a persuasive pervasive-based serious game approach to help people understand domestic energy usage and change their habits. It is characterized by real-time domestic energy consumption data, using smart energy monitor devices [MSS+11]. The application is characterized as a pervasive serious game because it is supposed to entertain and engage users, using an educational approach.

The main menu is characterized by an avatar that represents a house. As seen in Figures 4.1, 4.2 and 4.3, this house may assume different states depending on the current performance, i.e., the house is green if the current energy consumption is below average levels , if within normal levels the house assumes a yellow colour and finally, gets red if the energy consumption is higher than normal. The score is obtained through the evaluation of energy consumption and activities in competition mode. Those figures also presents an energetic scale according to the official energy efficiency rating from A (most efficient) to G (least efficient) that depends on the house's profile and current score.

Besides the single mode, the game also presents two secondary competition modes: combat and tournament. In the combat mode, the user is able to challenge any other player. The challenge consists on an environmental sustainability-based quiz to each user, and according to the result they will be awarded with a certain amount of points.

Unlike the combat mode, a tournament involves various users (between three and ten of them). It is created/organized by a single user that invites other users to participate, during a pre-determined period, usually one month, while the tournament classification is updated in real-time.

4.1.1 Projects Related to LEY

Besides LEY software (see chapter 4), there are more persuasive feedback systems related to domestic energy consumption-awareness. For instance, *Power Explorer* is a pervasive action-oriented multi-player game where the overall goal is to explore the household, learn about its electricity consuming devices and develop a positive attitude towards conserving electricity. However, it was designed for teenagers living at home with families and sharing their households with parents and siblings [BSG09]. *EnergyLife* is another system for households providing appliance-level data through a mobile application and feedback on the total energy consumption using an ambient display [JSG+09].

Although very similar to LEY, both applications do not implement any kind of personalization approach, however *EnergyLife* is currently studying that possibility.

4.2 Personalization Applied to LEY

Applications may differ a lot from each other, and some of them have more potential than others when considering the integration of personalization. Despite this condition, even simpler applications, such as LEY, can greatly benefit from the adoption of personalization.

The single-player basically consists on the main menu interface, which figures the current energy consumption, and also some secondary resources such as the statistics menu, where the user can consult usage statistics. The available statistics are the current energy score, the average energy consumption, number of played combats, number of rejected combats, number of played tournaments and number of rejected tournaments. The analysis of what LEY offers to its users results in several possibilities to collect different useful data. In order to use the application, a user needs to login, and this can be a good source of data, because users that constantly login have a different profile in comparison to users that login once a month, i.e., the former tend to be more interested and dedicated. But the number of logins may be not enough, because there may be users that are constantly logging in and logging out, and other users that rarely do so, but the latter might spend more time using LEY for each login session. This implies that the time spent logged in is also important in the user profile generation. Another important source of data is the number of times the user consults the statistics menu. If a user does that often, it probably implies that there is more dedication or interest on the user's performance.

Although the single-player mode provides data that can potentially be used for personalization, the multi-player mode overtakes this potential due to its interactive nature.

It is possible to collect the amount of combats and tournaments played by the user, the number of combats and tournaments a user has rejected, the category level of each user's opponents, how much time does a user take to complete a combat or a tournament and many other forms of data. This shows that LEY can provide quality data to used by a machine learning solution.

It is also important to know how LEY can be personalized in order to improve its users' experience when using the game. Initially, users probably take an interest in LEY because they have some environmental or economical consciousness. These concerns imply that users desire to optimize their usage habits in order to have a more energy-efficient house. Given the fact that for each user there is space for improvement, and that each user is different from each other, the ideal is that LEY learns how much each user is really interested in it, and what each user really desires from the application. After discussing the personalization possibilities, the members of the LEY/DEAP project decided to have four types of personalization to be created. Those are described in the following sub-sections. Some of those personalization instances are shown in Figures 4.1, 4.2 and 4.3. The final configuration file required by CAPE (see section 3.2) is presented in section C.

Title	Description
House Icon	The house icon varies between a fearful, defender and competitive image.
First Menu	The first screen to open up when logging in is the status, combat or tournament screen.
Status Background Neighbourhood	Background image varies according to the local weather and user contacts energy performance.
Alert Level Notifications	Mobile alert messages are personalized according to the user profile.

Table 4.1: Summary of personalization applied to LEY

4.2.1 House Icon

The central point of LEY is definitely the house icon that appears in the center of the main screen and it is probably the most-viewed element of the interface. This may result in an automatic association between the user and LEY, i.e., it can be considered as an avatar of the user's house. Assuming the user interprets the house (icon) as its own avatar it is interesting to make it more expressive. As it was referred previously, different users will have different interests and behaviour, which is why the house is a perfect candidate to express different personalities among users.

This personalization consists on the profiling of each user with one of three options:

fearful, *defender* and *competitive*. The fearful user profile is achieved when a user has rejected several competition proposals and does not make competition proposals. A user that is associated with a fearful level will observe a white flag next to the house icon in order to transmit the idea of "giving up". Just like the fearful user profile, a defender does not make many competition propositions, but on the other hand, if another user makes him/her a competition proposition, the defender will accept. This idea is represented by the usage of a blue medieval shield next to the house icon. Finally, the competitive user profile is used to represent users that regularly enter competition, either proposed to other users, or received from them. This profile is represented with the usage of a medieval sword next to the house icon. Figures 4.1 to 4.3 show those differences.

4.2.2 First Menu

When a user enters the LEY application s/he is faced with the authentication page, which is followed by the main application's window. This window is divided in three parts: status, combat and tournament. The user is able to slide along those three different windows as he/she wishes, but as it was referred previously, different users will have different interests. This means that some users would prefer to open LEY and go straight to the combat/status/tournament window.

With the analysis of how many combats and tournaments per unit of time, the user has done in the past, it is possible to define if s/he is more similar to a specific profile. This fact opens the door to the personalization of the first screen that is shown to each user, once LEY is opened. If a user frequently enters combats with other users, the combat window will be shown. If the user frequently enters tournaments with other users, the tournament window will be shown. Finally, if none of those situations occur, the house status window is presented to the user.

4.2.3 Status Background Neighbourhood

It has been mentioned before that CAPE can use external services such as meteorology services. Given a specific location it provides a very detailed description of the current weather conditions. This can be quite useful because the energy consumption of a user is also related to the weather conditions. For instance, during a snow storm in winter the users will tend to use warming devices to compensate the heat discrepancy, which results in an increase in energy consumption.

This personalization combines the usage of the current weather condition with how social a user is. The word *social* is used here in order to describe if a user uses LEY in a social way or non-social way. If a user is social, the landscape with vary according to the users energy consumption in comparison to the average energy consumption of the user's friends. The landscape, positioned in the screen's background can vary between flourishing, neutral or dry.

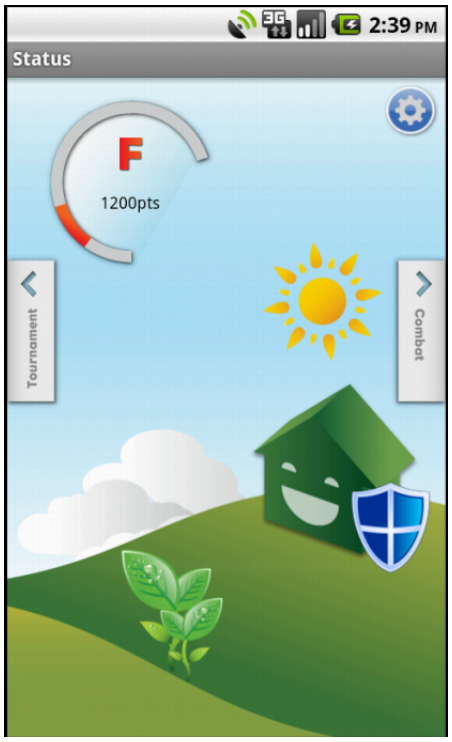


Figure 4.1: Sunny day, the house's instant energy consumption is below average and the user has defender profile

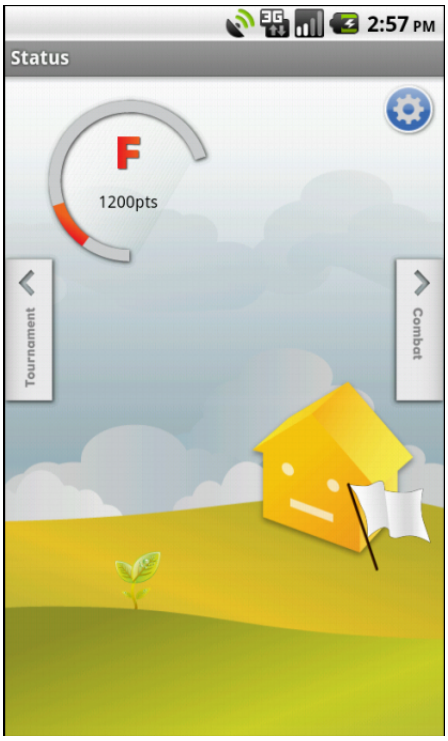


Figure 4.2: Cloudy day, the house's instant energy consumption is within average and the user has a fearful profile

Figure 4.3: Stormy day, the house's instant energy consumption is above average and the user has a competitive profile

4.2.4 Alert Level Notifications

LEY was implemented in an Android operating system environment, which means it is possible that a user receives messages/notifications, when a certain event occurs. Before the implementation of personalization, the application used to send regular alerts to the user every time the actual energy consumption exceeds a certain threshold. Although it is a good idea to warn the users when this happens, it is also interesting to personalize those alert messages according to the behaviour of each user.

This personalization uses the context segmentation approach in order to get more accurate results (see section 3.2). That data is derived from the user level (basic, intermediate and advanced user) and the level of competition (competitive and non-competitive user). The provided personalization options consist of a combination of the preceding personalization options.

Concerning the user level, a more advanced user tends to:

- Receive alert messages more often;
- Have a smaller threshold of energy consumption alert;
- Receive alerts accompanied by vibration, light works and sound from the mobile device;
- Receive a more detailed alert message, providing some additional data such as the difference between the current energy consumption and the current user's average;

There is also another type of alerts, which concerns to the competition level of a user. It consists in querying the date of the last competition the user was into, and comparing it with a certain interval of time that depends on the user's profile (for a competitive user, the interval is 2 days, otherwise the interval is 7 days). If the user was not in any competition for a period longer than the defined interval, then an alert is sent, informing him/her about this fact.

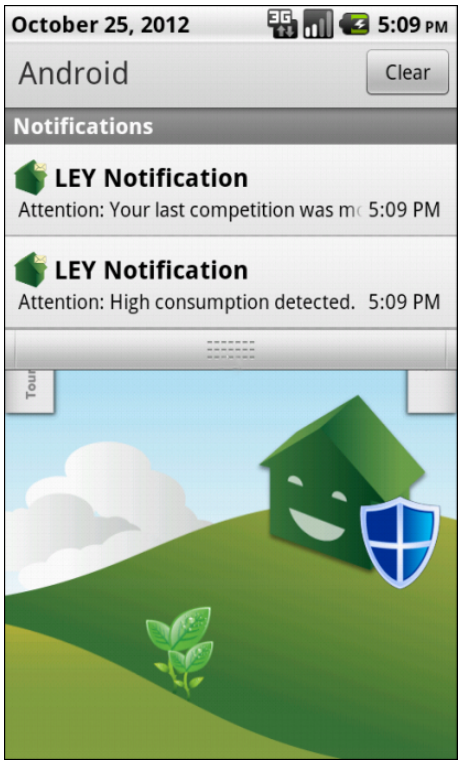


Figure 4.4: Two alert messages in the alert message box.

4.3 Discussion

CAPE was applied to LEY software by providing a set of personalization instances that enhance the application to a whole new level of dynamics. The developer only has to create an XML file that describes the application to be personalized, and then adapt the application's source code. Through HTTP requests, LEY was able to submit data to the web server where CAPE was hosted, and also send requests of user profiles for each different personalization type. With those profiles, it is easy to introduce personalization into the application, because the type of user that is using LEY is already known.

5

Evaluation

In this chapter it is presented an evaluation of the configuration process each developer must take in order to register an application within CAPE. Since this procedure is supposed to be executed by application developers, it was assured that every participating user in the following tests were familiar with programming, and in particular with XML. Before the tests were executed, each user was faced with a brief presentation of what CAPE offers, and the pre-requisites that should be fulfilled in order to benefit from CAPE's personalization (shown in section 3.2). While the first phase of tests was applied to LEY software, the second phase presented a more general personalization approach because CAPE was applied to each developer's own chosen application.

5.1 Tests Based on LEY

The first phase of evaluation tests was based on the LEY software, presented in chapter 4.

5.1.1 Evaluation

In the beginning of the evaluation, both CAPE and LEY were introduced in detail to a set of ten developers, all of them from different areas of the computer science domain (Figure C.5), with different levels of experience (Figure C.6) and with an age that varied between 21 and 26. After that introduction, and being assured that each user understood the objectives, motivation and inner workings of LEY and CAPE, it was proposed that each user would try to configure an XML file in order to introduce personalization into LEY. To each user was given a different personalization to implement, being assured that each

different personalization had a similar level of difficulty, implementation-wise. Each user understood the task at hand and was confronted with the creation of one personalization example and multiple parameters and resources, to support that personalization. They were told to imagine themselves as LEY developers, which means that they were offered the freedom to use or create any data resource they thought was obtainable from LEY's usage.

Although some users took more time than others, they were all able to implement the given task. It was observed that during the tests, the concept of context segmentation was not immediately understood, but when the notion was assimilated they liked the concept. Another interesting point was the fact that the users did not perceive the data normalization problem. This issue happens when a parameter formula is structured in a way that results in data that is not balanced in the long term. For instance, if a parameter formula only measures the number of clicks in a certain menu, long time users will tend to have a larger value in comparison to newer users. The solution to this issue consists in creating an expression that represents the frequency of clicks per minute of measured log-in time.

5.1.2 Survey

Following the test phase, each user filled a survey (presented in section C.1), which enabled the attainment of several conclusions.

Every user indicated that they had never used any tools or approach to integrate personalization into their applications, as shown in Figure C.7. This shows that they have no means to compare CAPE with third party personalization techniques, and that it was the first time they had to think about personalizing an application.

Figure C.8 shows that it is clear that all users understood the connection between the concept of personalization-parameter-resource. Figure C.9 shows a tendency for the users to find the configuration methodology reasonably easy. On the same note, Figure C.10 shows that users rated their practical personalization task applied to LEY as reasonably easy, although the results were not as clear as the ones from Figure C.9, with a slightly higher average regarding difficulty level. One very interesting conclusion is the fact that users strongly accepted the notion of context in order to refine their data. This acceptance is clearly demonstrated in the results of Figure C.11. During the experiments they referred that the usage of context offers a great potential for some specific applications.

Regarding the adoption of other external services and context types the results (Figure C.12) are unclear because some users thought that the current available services and context types are quite limiting, and other users thought that there was not any other service that could improve the personalization results. Among their suggestions there is the adoption of the mobile camera to recognize the current user's surroundings. If applicable, it could be interesting to take into account different energy fares a user might

have, which vary according to daytime, and also the usage of GPS coordinates to detect if a user is in any particular place such as his/her workplace, home or other places he/she might frequent in a regular way.

It is also interesting to know what developers think about the main limitations in this type of configuration procedure. In order to known which limitations they consider the most critical, the survey figured three questions concerning the steepness of the learning curve, lack of expressiveness in the configuration model and the level of complexity regarding the configuration process in general. These questions were used in order to obtain information about the priorities of each user regarding this type of procedure.

Figure C.13 shows that users consider the learning curve steepness as a reasonable limitation when configuring an application, implying that this issue has some influence on their experience. Figure C.14 presents the results of how limiting users consider the lack of expressiveness in the configuration process. It is possible to conclude from those results that users believe this is possibly an important limitation in this kind of procedure. Figure C.15 shows that users consider the level of complexity in the configuration process as somewhat irrelevant.

From the previous three results it is possible to conclude that developers prefer a configuration process that can be learned fairly quickly and can be strongly expressive in order to provide the proper personalization, regardless of the level of complexity it presents. Finally, when faced with how much they would like to use CAPE in the future, Figure C.16 shows that there is a clear interest in using it, which greatly indicates the users were pleased with their experience. Regarding the comments, doubts and suggestions question, no information was written.

5.2 Tests Based on Other Applications

In this second phase of tests, instead of suggesting personalization that would apply to LEY, users were told to choose any application they had developed in the past or were currently developing. Next, they were asked to choose one possible personalization that could benefit their application and then implement the configuration of that personalization, following CAPE's defined structure requirements in XML.

This test involved five computer programmers with ages between 24 and 33, and different levels of experience (Figure C.18). Unlike the previous phase of tests, most users in this second phase work in the Human Computer Interfaces and MultiModal Systems area, which implies some experience with high-level applications. The distribution can be seen in Figure C.17.

The applications each user chose, are very different from each other. Applications concerning sports news, multi-player fighting games, augmented reality, are some examples of applications that benefited from the integration of personalization. By evaluating the reaction of those users, when confronted with the need to personalize their chosen

application, it was clear that the task should be thought over for some time. Nevertheless, after a few minutes, the users were able to imagine situations that could benefit the application, when applying personalization using CAPE. It was observed that users were very enthusiastic concerning the possibilities of CAPE, and tried to use in their configuration most techniques CAPE has to offer, with context segmentation having a particular focus. It is also possible to point out that most users did not immediately perceive the data normalization problem, referred in the first phase of tests.

After the implementation part, this second phase of tests followed the same approach as the first one, consisting on filling the same survey as in the previous phase of tests (presented in section C.1), which provided some interesting conclusions.

Following the same direction as the first phase of tests, Figure C.19 shows that no user ever used any tools or approach to integrate personalization into their applications.

Most users understood the concept of personalization-parameter-resource. Figure C.20 shows that a single user answered negatively to this question, and justified the answer by saying that a graphic user interface would probably make that connection more intuitive.

Figure C.21 clearly shows that users found the theoretical configuration methodology fairly easy. On the same note, Figure C.22 proves that users found the practical experience with the given task, to be easy to implement, although the results were not as clear as the ones from Figure C.21, because the former show a slightly higher average regarding difficulty level.

The previous phase of tests showed that users strongly accepted the notion of 'context' as a useful source of additional information. This second phase supports the same conclusion, as Figure C.23 shows.

When asked about the lack of types of external services and context types on CAPE, the results were not conclusive. On the first phase of tests, there was a 50% division between users who thought there should be more types, and 50% who thought the given types are enough. Unfortunately, on this second phase of tests, the answer is still not clear as Figure C.24 shows. Although there is a small tendency for a positive response, the amount of tested users is not enough to draw a strong conclusion.

Regarding the tested users' opinion concerning possible limitations in a personalization configuration process, the results were very similar to the presented ones in the first phase of tests. Figure C.25 shows that users consider the learning curve steepness as a reasonable limitation when configuring an application, implying that this issue has some influence on their experience. In comparison to the results in the previous phase of tests, the graph presents a slightly lower average concerning the limitation's severity.

Regarding how limiting is the lack of expressiveness, users' answers show that it can be a limitation to consider. Figure C.26 shows that result, which is similar to the results in the previous phase of tests. Finally, Figure C.27 follows the same tendency as in the previous phase of tests, showing that users consider the level of complexity in the configuration process as somewhat irrelevant.

It is shown that the three results, concerning possible limitations in a configuration methodology, follow the same trend as the results on the previous phase of tests, i.e., it is possible to conclude that developers prefer a configuration process that can be learned fairly quickly and can be strongly expressive in order to provide the proper personalization, regardless of the level of complexity it presents.

Figure C.28 clear shows that the tested users would be interested in using CAPE for their future developments.

Regarding the comments, doubts and suggestions question, some users expressed satisfaction regarding the usage of XML to configure their applications. It was also referred that CAPE needs more machine learning technologies, for different applications that focus more on the recommendation of items.

6

Conclusions and Future Work

This chapter presents an analysis of the work produced under the scope of this dissertation, as well as solutions and ideas for the future work that extends the contribution presented in this document.

6.1 Conclusions

This work presents a Context-Aware Personalization Environment (CAPE) that is used to generate user profiles in order to provide personalization to mobile applications. User profiles are generated by a machine learning module using K-means clustering algorithm, that resorts to interaction stream data, user preferences and context data, obtained when the personalization request is made. In order to be fully operational, CAPE offers an API (Application Programming Interface) to application developers, that allows them to submit interaction stream data updates, user preferences updates, creation of new users and, of course, personalization requests.

CAPE was developed to fulfill the demand for adaptation and personalization that pervasive and mobile devices require, in order to weaken their natural technological limitations such as small screens and limited processing power.

The developed prototype was applied and tested on LEY (Less energy Empowers You) software, which is a persuasive mobile *serious* game approach to help people understand domestic energy usage in order to change their negative habits. LEY benefited from different types of personalization such as the level of alert messages it sends to its users, graphic interface modifications and personalization and dynamic menu behaviour.

Besides the LEY case-study, several tests were made with external software developers, in order to learn their opinion about what CAPE has to offer them. Those tests consisted on the creation of a configuration file that would allow an application to benefit from personalization. Each developer received an explanation concerning the structure of the desired configuration file, and was also informed about LEY in a detailed way. After that, the developers were offered a personalization idea and asked to implement it. The results showed that every developer was able to fulfil the task and most of them reported they would like to use CAPE in the future for their own purposes.

Besides the developed work in this dissertation, a 2-page work in progress was accepted at the ACM Ubicomp 2012 conference. Ubicomp is described as the premier outlet for novel research contributions that advance the state of the art in the design, development, deployment, evaluation and understanding of ubiquitous computing systems.

This work was also sent to an idea contest promoted by Fraunhofer AICOS company, named Fraunhofer Portugal Challenge. The Challenge consists in awarding the best ideas based in graduation thesis that were developed having *Research of Practical Utility* in mind, i.e., ideas based on thesis concepts that clearly demonstrate a concern with the direct applicability of its results in Industry. The submission successfully reached the 2nd phase of the challenge.

6.2 Future Work

In its current state, CAPE is a prototype that is fully functional and ready to be applied to any application, but there are still several aspects that can improve its functioning.

The prototype only implements one type of machine learning algorithm, named K-means clustering algorithm. Although K-means is not a recent algorithm, it provides very good results, it is simple and widely tested. K-means also has some limitations in some specific circumstances, which is why it is important to create other clustering algorithms that could be used interchangeably in the personalization process according to each specific situation. The adoption of a hierarchical clustering algorithm could prove extremely useful for situations where the number of clusters/parameter options is unknown.

The K-means implementation in CAPE only uses one distance metric to create a level of differentiation between two points. Depending on the data this can prove to be a limitation, specially in cases where multiple formulas are being used within a single parameter. Implementations such as Tanimoto distance (see section 2.1.3) can be quite interesting to overcome this limitation because although they consider the direct distance between two points just like the normal Euclidean distance, they also use the angle between those points, by creating a vector abstraction with them.

CAPE also does not provide any mechanism to test the effectiveness of the provided personalization on an application, which implies the need to create a quality testing module for the developer, so that s/he can figure what should be the best approach to follow.

It was also mentioned during the tests that the prototype could benefit from the implementation of more types of updates on data such as subtraction and decrement operations. Those tests also mentioned that the prototype could use more context resource types and external services in order to make the configuration process more wide in terms of expressiveness. Some suggested examples are the connection to:

- Traffic web-service that informs the current state of traffic in a given location;
- Meteorology service that provides current levels of pollution in the atmosphere;
- Location service that tracks familiar places where the user tends to go.

The participants in the tests also pointed out, that the development of a graphic user interface to register and configure an application on CAPE, would benefit the pre-personalization process.

Bibliography

[Alp04] Ethem Alpaydin. *Introduction to Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2004.

[BBC97] P. J. Brown, J. D. Bovey, and X. Chen. Context-aware Applications: from the Laboratory to the Marketplace. *IEEE Personal Communications*, 4(5):58–64, 1997.

[Ber02] Pavel Berkhin. Survey Of Clustering Data Mining Techniques. Technical report, Accrue Software, Inc., Fremont, Canada, 2002.

[BSG09] M. Bang, M. Svahn, and Anton Gustafsson. Persuasive Design of a Mobile Energy Conservation Game with Direct Feedback and Social Cues. *Breaking New Ground Innovation in Games Play Practice and Theory Proceedings of DiGRA 2009*, 2009.

[Bur02] R. Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12, November 2002.

[DAS01] Anind K. Dey, Gregory D. Abowd, and Daniel Salber. A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-aware Applications. *Hum.-Comput. Interact.*, 16(2):97–166, December 2001.

[DPDM09] T. De Pessemier, T. Deryckere, and L. Martens. Context-aware Recommendations for User-generated Content on a Social Network Site. In *Proceedings of the seventh european conference on European interactive television conference*, EuroITV '09, pages 133–136. ACM, 2009.

[GKvR07] Patrick J.F. Groenen, Uzay Kaymak, and Joost van Rosmalen. *Fuzzy Clustering with Minkowski Distance Functions*, pages 53–68. John Wiley & Sons, Ltd, 2007.

[JD88] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1988.

[JMF99] A. K. Jain, M. N. Murty, and P. J. Flynn. Data Clustering: a Review. *ACM Comput. Surv.*, 31(3):264–323, September 1999.

[JSG+09] Giulio Jacucci, Anna Spagnolli, Luciano Gamberini, Alessandro Chalambalakis, Christoffer Björkskog, Massimo Bertoncini, Carin Torstensson, and Pasquale Monti. Designing Effective Feedback of Electricity Consumption for Mobile User Interfaces. *PsychNology Journal*, 7(3):265–289, 2009.

[JTD06] Ivar Jørstad, Do Van Thanh, and Schahram Dustdar. Personalisation of Next Generation Mobile Services. In *UMICS'06*, 2006.

[Kur07] M. Kurze. Personalization in Multimodal Interfaces. In *Proceedings of the 2007 workshop on Tagging, mining and retrieval of human related activity information*, TMR '07, pages 23–26, New York, NY, USA, 2007. ACM.

[Mad12] Rui Neves Madeira. Personalization in Pervasive Spaces Towards Smart Interactions Design. In *PerCom Workshops*, pages 548–549, 2012.

[MSS+11] Rui Neves Madeira, Andre Silva, Catarina Santos, Bárbara Teixeira, Teresa Romao, Eduardo Dias, and Nuno Correia. LEY!: persuasive pervasive gaming on domestic energy consumption-awareness. In *Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology*, ACE '11, pages 72:1–72:2, New York, NY, USA, 2011. ACM.

[MVC12] Rui Neves Madeira, André Vieira, and Nuno Correia. Personalization of an Energy Awareness Pervasive Game. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, pages 619–620. ACM, 2012.

[NH12] W. Nordstrom and J. Hakansson. Finding Clusters of Similar Artists - Analysis of DBSCAN and K-means Clustering. Technical report, Royal Institute of Technology, 2012.

[NHL07] Petteri Nurmi, Marja Hassinen, and Kun Chang Lee. A Comparative Analysis of Personalization Techniques for a Mobile Application. In *Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops - Volume 02*, AINAW '07, pages 270–275, Washington, DC, USA, 2007. IEEE Computer Society.

[NM11] Pooja Batra Nagpal and Priyanka Ahlawat Mann. Comparative Study of Density based Clustering Algorithms. *International Journal of Computer Applications*, 27(11):44–47, August 2011. Published by Foundation of Computer Science, New York, USA.

[OADF11] S. Owen, R. Anil, T. Dunning, and E. Friedman. *Mahout in Action*. Manning Publications, 2011.

[PAS06] Christoforos Panayiotou, Maria Andreou, and George Samaras. Using Time and Activity in Personalization for the Mobile User. In *Proceedings of the 5th ACM international workshop on Data engineering for wireless and mobile access*, MobiDE '06, pages 87–90, New York, NY, USA, 2006. ACM.

[PS04] Christoforos Panayiotou and George Samaras. mPERSONA: personalized portals for the wireless user: An agent approach. *Mob. Netw. Appl.*, 9:663–677, December 2004.

[PTG08] Cosimo Palmisano, Alexander Tuzhilin, and Michele Gorgoglione. Using Context to Improve Predictive Modeling of Customers in Personalization Applications. *IEEE Trans. on Knowl. and Data Eng.*, 20:1535–1549, November 2008.

[Ree00] George Reese. *Database Programming with JDBC and Java, Second Edition*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2nd edition, 2000.

[RRSK11] F. Ricci, L. Rokach, B. Shapira, and P. B. Kantor. *Recommender Systems Handbook*. Springer, 2011.

[RS10] P. Rai and S. Singh. A Survey of Clustering Techniques. *International Journal of Computer Applications (0975-8887)*, 7(12), October 2010.

[SBL12] N. Sharma, A. Bajpai, and R. Litoriya. Comparison the Various Clustering Algorithms of Weka Tools. *International Journal of Emerging Technology and Advanced Engineering (2250-2459)*, 2(5), May 2012.

[TSK01] Ben Taskar, Eran Segal, and Daphne Koller. Probabilistic classification and clustering in relational data. In *Proceedings of the 17th international joint conference on Artificial intelligence - Volume 2*, IJCAI'01, pages 870–876, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[TSK05] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[VPK+11] Athanasios S. Voulodimos, Charalampos Z. Patrikakis, Pantelis N. Karamolegkos, Anastasios D. Doulamis, and Emmanuel S. Sardis. Employing Clustering Algorithms to Create User Groups for Personalized Context Aware Services Provision. In *Proceedings of the 2011 ACM workshop on Social and behavioural networked media access*, SBNMA '11, pages 33–38, New York, NY, USA, 2011. ACM.

[WDFLP08] Diana Weiß, Markus Duchon, Florian Fuchs, and Claudia Linnhoff-Popien. Context-aware Personalization for Mobile Multimedia Services. In *Proceedings of the 6th International Conference on Advances in Mobile Computing and Multimedia*, MoMM '08, pages 267–271, New York, NY, USA, 2008. ACM.

[WF05] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[WS02] M. Wallace and G. Stamou. Towards a Context Aware Mining of User Interests for Consumption of Multimedia Documents. In *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME)*, 2002.

[WS07] Wolfgang Woerndl and Johann Schlichter. *Introducing Context Into Recommender Systems*, pages 138–140. 2007.

[WUS09] R. Wetzker, W. Umbrath, and A. Said. A Hybrid Approach to Item Recommendation in Folksonomies. In *Proceedings of the WSDM '09 Workshop on Exploiting Semantic Annotations in Information Retrieval*, ESAIR '09, pages 25–29, New York, NY, USA, 2009. ACM.

[XEKS98] Xiaowei Xu, Martin Ester, Hans-Peter Kriegel, and Jörg Sander. A distribution-based clustering algorithm for mining in large spatial databases. In *Proceedings of the Fourteenth International Conference on Data Engineering*, ICDE '98, pages 324–331, Washington, DC, USA, 1998. IEEE Computer Society.

[YW08] Junsong Yuan and Ying Wu. Context-aware Clustering. *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 0:1–8, 2008.

[Zha10] Yagang Zhang. *New Advances in Machine Learning*. InTech, 2010.

[ZJL12] W. Zhou, H. Jin, and Y. Liu. Community Discovery and Profiling with Social Messages. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '12, pages 388–396, New York, NY, USA, 2012. ACM.

Appendix A - Personalization Data Model

The following diagram represents the complete personalization data model used in this framework. This diagram represents the combination of diagram 3.10, 3.11 and 3.12.

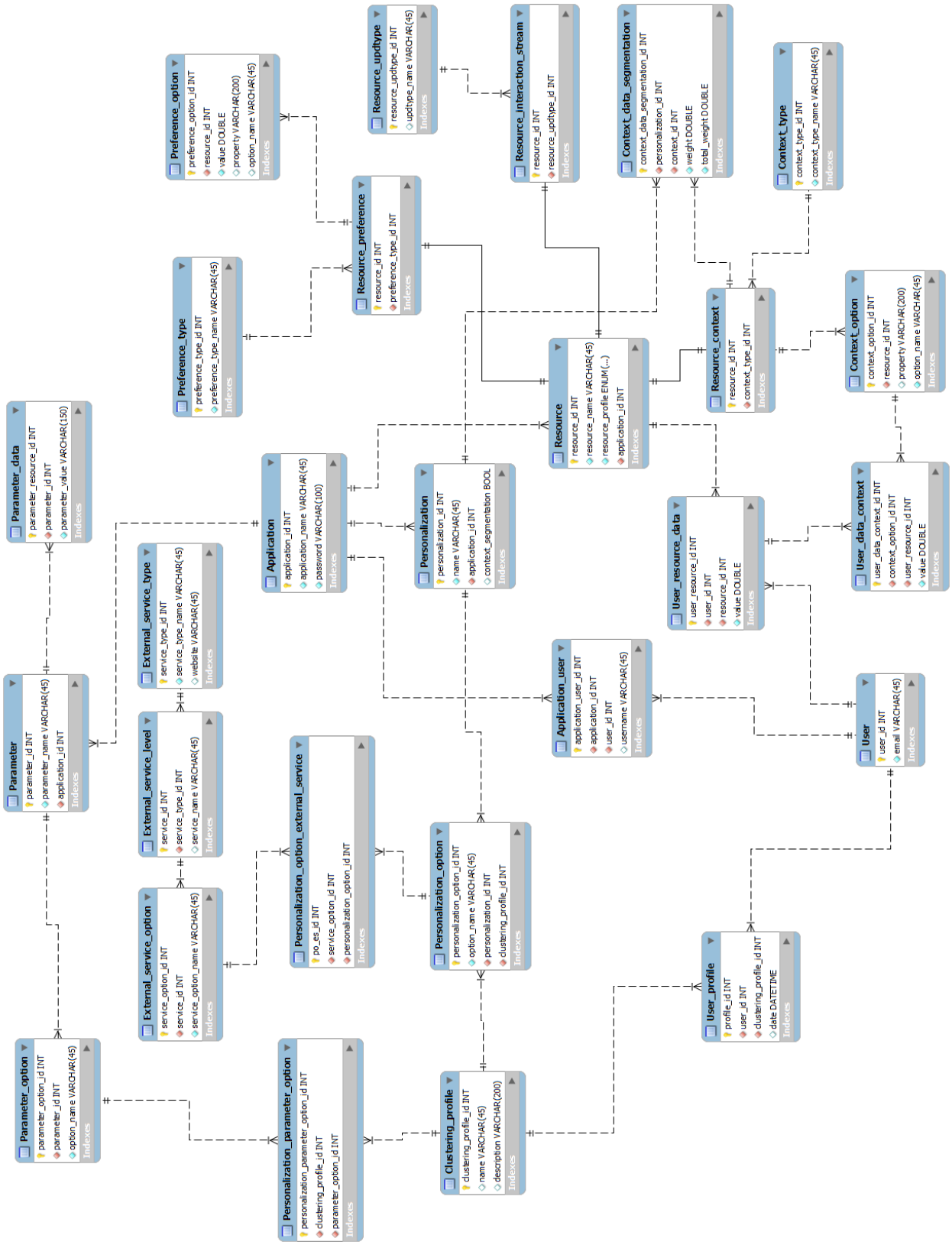


Figure A.1: The complete data model

B

Appendix B - LEY Configuration File

```
<?xml version='1.0' standalone='yes'?>
<config>
  <database>
    <name>ley</name>
    <password>applicationpassword</password>
  </database>

  <personalizations>
     <personalization>
    <name>first screen</name>
    <type>clustering</type>
    <personalization_option name="status">
      <parameter_option parameter_name="combat_profile">non combat</parameter_option>
      <parameter_option parameter_name="tournament_profile">non tournament</
          parameter_option>
    </personalization_option>
    <personalization_option name="combat">
      <parameter_option parameter_name="combat_profile">combat</parameter_option>
      <parameter_option parameter_name="tournament_profile">non tournament</
          parameter_option>
    </personalization_option>
    <personalization_option name="tournament">
      <parameter_option parameter_name="combat_profile">non combat</parameter_option>
      <parameter_option parameter_name="tournament_profile">tournament</parameter_option
          >
    </personalization_option>
     </personalization>

     <personalization>
    <name>alert level</name>
    <type>clustering</type>
    <context>
      <total_context_weight>0.25</total_context_weight>
      <context_resource>
```

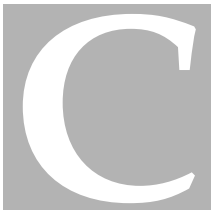
```
32          <name>routineHours</name>
33          <weight>0.6</weight>
34        </context_resource>
35        <context_resource>
36          <name>currentTemp</name>
37          <weight>0.4</weight>
38        </context_resource>
39      </context>
40      <personalization_option name="single_basic">
41        <parameter_option parameter_name="user level">basic</parameter_option>
42        <parameter_option parameter_name="competition_level">non competitive</
              parameter_option>
43      </personalization_option>
44      <personalization_option name="single_advanced">
45        <parameter_option parameter_name="user level">advanced</parameter_option>
46        <parameter_option parameter_name="competition_level">non competitive</
              parameter_option>
47      </personalization_option>
48      <personalization_option name="multi_basic">
49        <parameter_option parameter_name="user level">basic</parameter_option>
50        <parameter_option parameter_name="competition_level">competitive</parameter_option
              >
51      </personalization_option>
52      <personalization_option name="multi_advanced">
53        <parameter_option parameter_name="user level">advanced</parameter_option>
54        <parameter_option parameter_name="competition_level">competitive</parameter_option
              >
55      </personalization_option>
56       </personalization>
57
58       <personalization>
59      <name>house image</name>
60      <type>clustering</type>
61      <personalization_option name="fearful">
62        <parameter_option parameter_name="fearful_level">fearful</parameter_option>
63        <parameter_option parameter_name="competition_level">non competitive</
              parameter_option>
64      </personalization_option>
65      <personalization_option name="fearful">
66        <parameter_option parameter_name="fearful_level">fearful</parameter_option>
67        <parameter_option parameter_name="competition_level">competitive</parameter_option
              >
68      </personalization_option>
69      <personalization_option name="defender">
70        <parameter_option parameter_name="fearful_level">non fearful</parameter_option>
71        <parameter_option parameter_name="competition_level">non competitive</
              parameter_option>
72      </personalization_option>
73      <personalization_option name="competitive">
74        <parameter_option parameter_name="fearful_level">non fearful</parameter_option>
75        <parameter_option parameter_name="competition_level">competitive</parameter_option
              >
76      </personalization_option>
77       </personalization>
78
79       <personalization>
80      <name>status background neighbourhood</name>
81      <type>clustering</type>
```

```
82      <personalization_option name="isolated sun">
83        <parameter_option parameter_name="competition_level">non competitive</
              parameter_option>
84        <external_service service_name="weather">sun</external_service>
85      </personalization_option>
86      <personalization_option name="neighbourhood sun">
87        <parameter_option parameter_name="competition_level">competitive</parameter_option
              >
88        <external_service service_name="weather">sun</external_service>
89      </personalization_option>
90      <personalization_option name="isolated clouds">
91        <parameter_option parameter_name="competition_level">non competitive</
              parameter_option>
92        <external_service service_name="weather">clouds</external_service>
93      </personalization_option>
94      <personalization_option name="neighbourhood clouds">
95        <parameter_option parameter_name="competition_level">competitive</parameter_option
              >
96        <external_service service_name="weather">clouds</external_service>
97      </personalization_option>
98      <personalization_option name="isolated rain">
99        <parameter_option parameter_name="competition_level">non competitive</
              parameter_option>
100       <external_service service_name="weather">rain</external_service>
101     </personalization_option>
102     <personalization_option name="neighbourhood rain">
103       <parameter_option parameter_name="competition_level">competitive</parameter_option
              >
104       <external_service service_name="weather">rain</external_service>
105     </personalization_option>
106      </personalization>
107   </personalizations>
108
109   <parameters>
110      <parameter>
111     <name>combat_profile</name>
112     <option>non combat</option>
113     <option>combat</option>
114     <data>
115         <data_value>numberCombats/loginTime</data_value>
116     </data>
117      </parameter>
118
119      <parameter>
120     <name>tournament_profile</name>
121     <option>non tournament</option>
122     <option>tournament</option>
123     <data>
124         <data_value>numberTournaments/loginTime</data_value>
125     </data>
126      </parameter>
127
128      <parameter>
129     <name>user level</name>
130     <option>basic</option>
131     <option>advanced</option>
132     <data>
133         <data_value>(0.8*loginTime+0.2*statistics)*question1</data_value>
```

```
134     </data>
135      </parameter>
136
137      <parameter>
138     <name>competition_level</name>
139     <option>non competitive</option>
140     <option>competitive</option>
141     <data>
142         <data_value>0.35*(numberCombats/loginTime)+0.65*(numberTournaments/loginTime)</
                data_value>
143     </data>
144      </parameter>
145
146      <parameter>
147     <name>fearful_level</name>
148     <option>non fearful</option>
149     <option>fearful</option>
150     <data>
151         <data_value>0.35*(numberRejectedCombats/(numberCombats/loginTime))+0.65*(
                numberRejectedTournaments/(numberTournaments/loginTime))</data_value>
152     </data>
153      </parameter>
154   </parameters>
155
156   <external_services>
157     <service>
158       <name>weather</name>
159       <type>weather_basic</type>
160       <default>sun</default>
161       </service>
162   </external_services>
163
164   <resources>
165     <resource_appdata>
166       <name>login</name>
167       <upd_type>increment</upd_type>
168     </resource_appdata>
169     <resource_appdata>
170       <name>loginTime</name>
171       <upd_type>sum</upd_type>
172     </resource_appdata>
173     <resource_appdata>
174       <name>statistics</name>
175       <upd_type>increment</upd_type>
176     </resource_appdata>
177     <resource_appdata>
178       <name>numberCombats</name>
179       <upd_type>increment</upd_type>
180     </resource_appdata>
181     <resource_appdata>
182       <name>numberTournaments</name>
183       <upd_type>increment</upd_type>
184     </resource_appdata>
185     <resource_appdata>
186       <name>numberRejectedCombats</name>
187       <upd_type>increment</upd_type>
188     </resource_appdata>
189     <resource_appdata>
```

```
190         <name>numberRejectedTournaments</name>
191         <upd_type>increment</upd_type>
192       </resource_appdata>
193
194       <resource_context>
195         <name>routineHours</name>
196         <type>hour_interval</type>
197         <option>
198           <name>businessHours</name>
199           <property>09:00-17:00</property>
200         </option>
201         <option>
202           <name>nonBusinessHours</name>
203           <property>17:00-09:00</property>
204         </option>
205       </resource_context>
206       <resource_context>
207         <name>currentTemp</name>
208         <type>temperature</type>
209         <option>
210           <name>Warm</name>
211           <property>21-40</property>
212         </option>
213         <option>
214           <name>Cold</name>
215           <property>0-21</property>
216         </option>
217       </resource_context>
218
219       <resource_preference>
220         <name>question1</name>
221         <type>question_MC</type>
222         <option>
223           <name>q1radio1</name>
224           <value>0.8</value>
225         </option>
226         <option>
227           <name>q1radio2</name>
228           <value>0.9</value>
229         </option>
230         <option default="true">
231           <name>q1radio3</name>
232           <value>1</value>
233         </option>
234         <option>
235           <name>q1radio4</name>
236           <value>1.1</value>
237         </option>
238         <option>
239           <name>q1radio5</name>
240           <value>1.2</value>
241         </option>
242       </resource_preference>
243
244     </resources>
245 </config>
```

Listing B.1: XML Configuration File

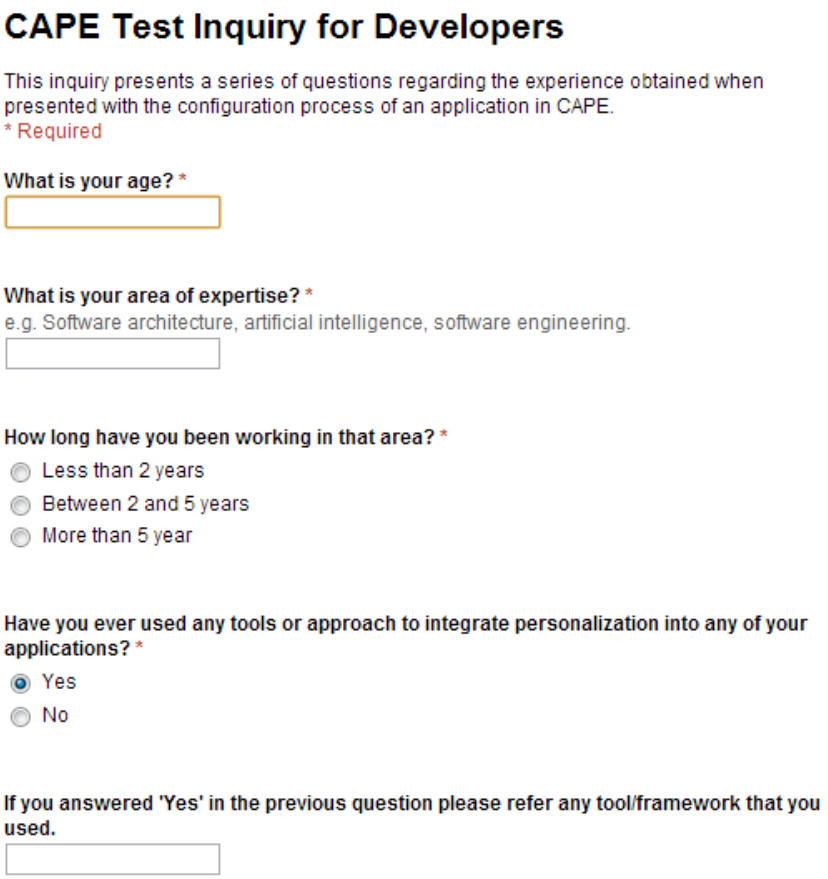


Appendix C - Evaluation

This appendix presents both types of tests that were applied to CAPE.

C.1 Tests Applied to LEY - Survey

The following three figures present the survey every user had to fill after taking the configuration test applied to LEY.



CAPE Test Inquiry for Developers

This inquiry presents a series of questions regarding the experience obtained when presented with the configuration process of an application in CAPE.
* Required

What is your age? *

What is your area of expertise? *
e.g. Software architecture, artificial intelligence, software engineering.

How long have you been working in that area? *
- Less than 2 years
- Between 2 and 5 years
- More than 5 year

Have you ever used any tools or approach to integrate personalization into any of your applications? *
- Yes
- No

If you answered 'Yes' in the previous question please refer any tool/framework that you used.

Figure C.1: Inquiry applied to LEY - part 1

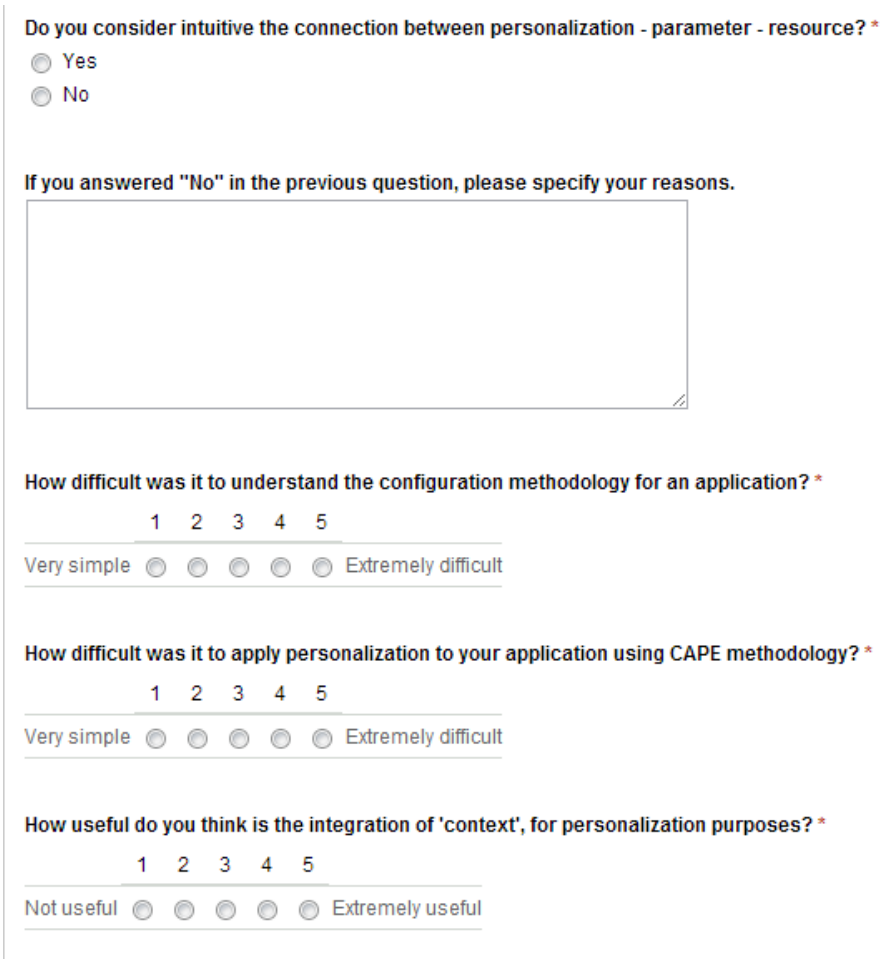


Figure C.2: Inquiry applied to LEY - part 2

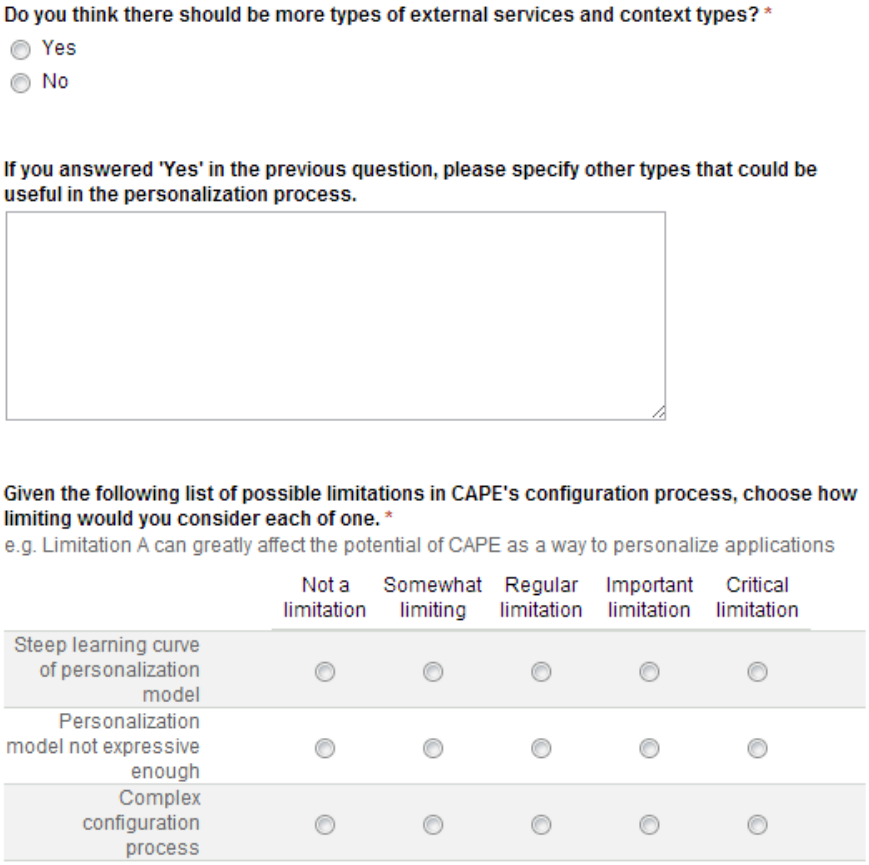


Figure C.3: Inquiry applied to LEY - part 3

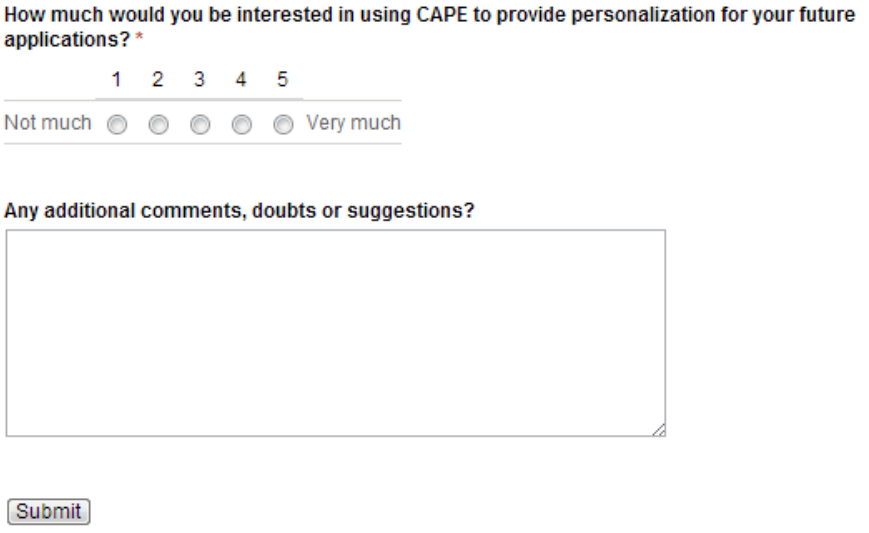


Figure C.4: Inquiry applied to LEY - part 4

C.2 Tests Applied to LEY - Results

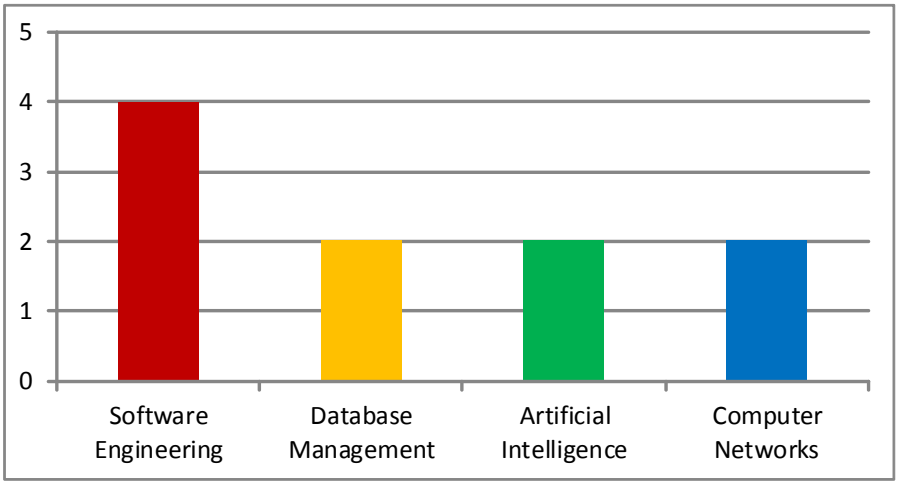


Figure C.5: What is your area of expertise?

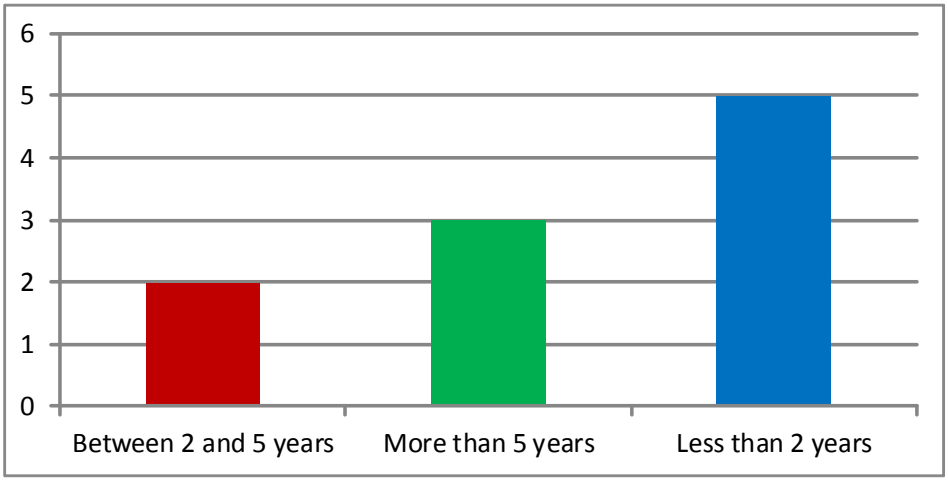


Figure C.6: For how long have you been working in that area?

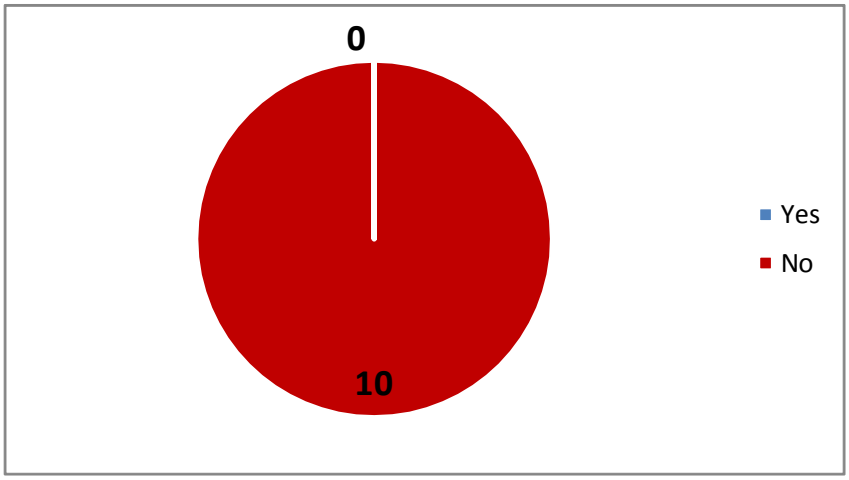


Figure C.7: Have you ever used any tools or approach to integrate personalization into any of your applications?

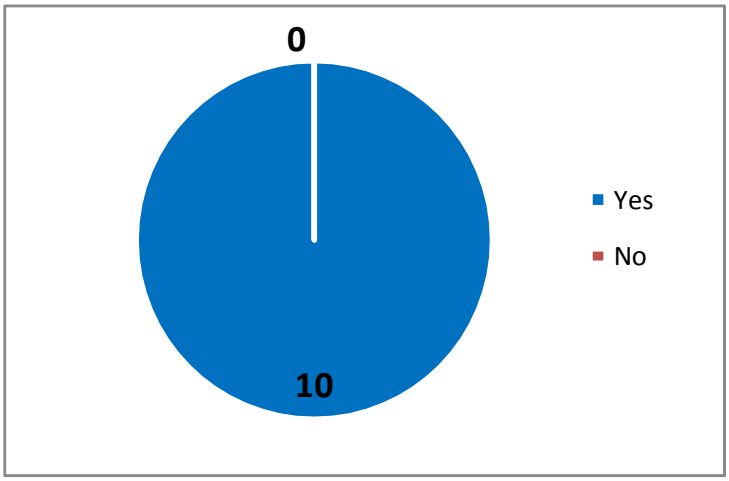


Figure C.8: Do you consider intuitive the connection between personalization - parameter - resource?

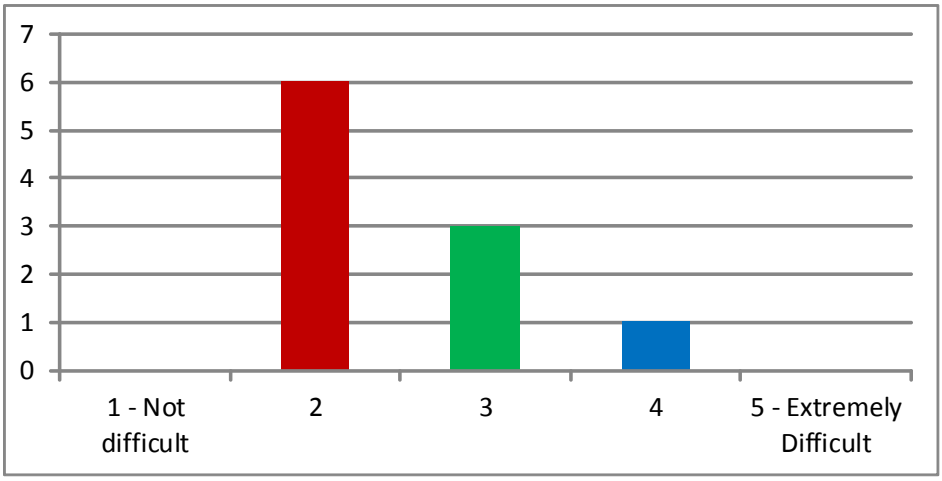


Figure C.9: How difficult was it to understand the configuration methodology for an application?

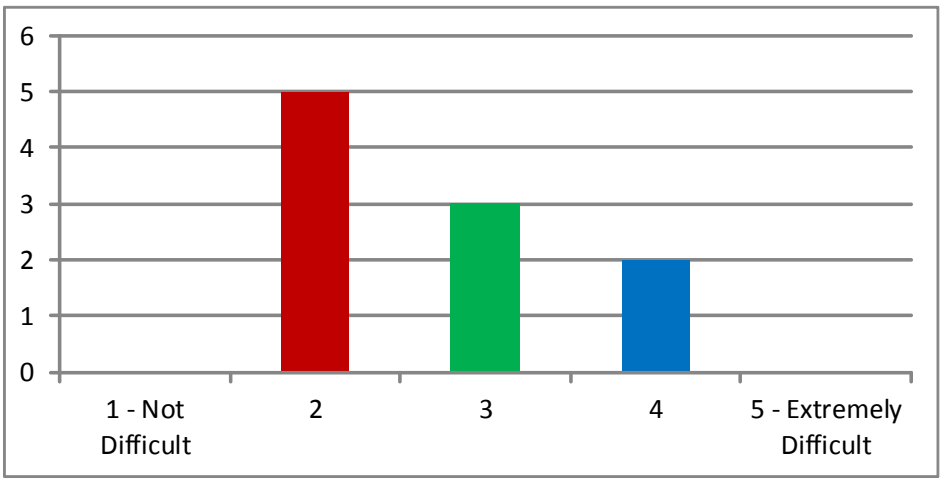


Figure C.10: How difficult was it to apply personalization to your application using CAPE methodology?

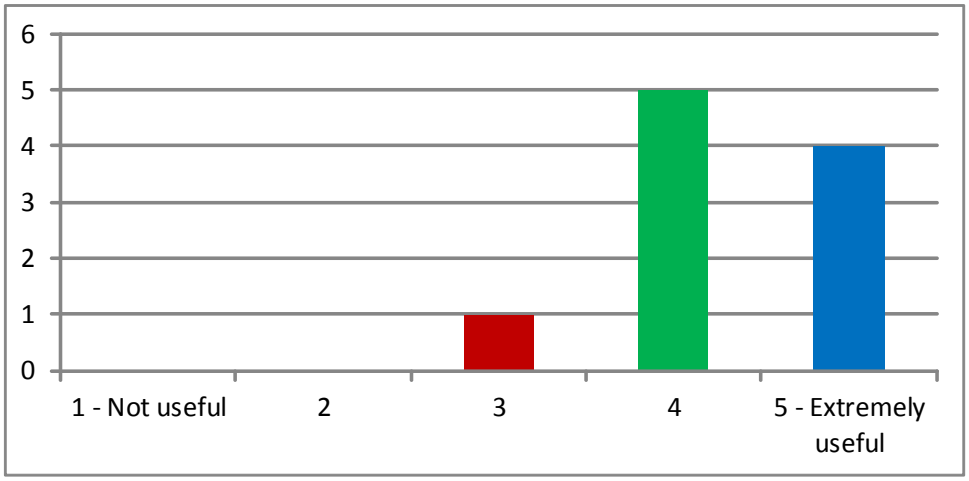


Figure C.11: How useful do you think is the notion of 'context', for personalization purposes?

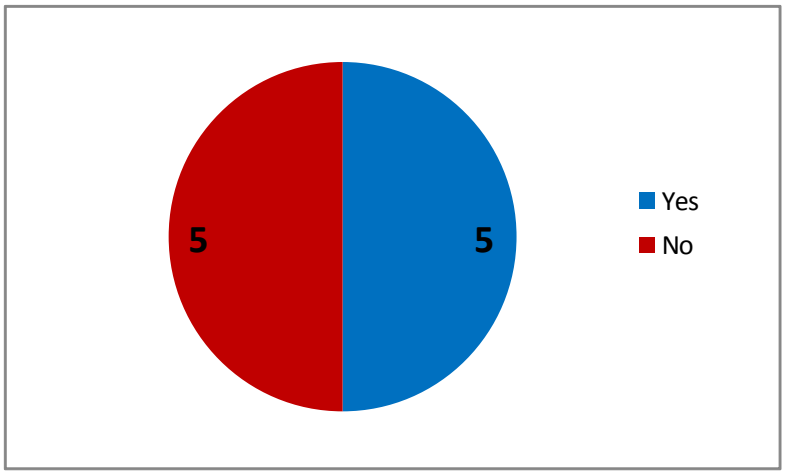


Figure C.12: Do you think there should be more types of external services?

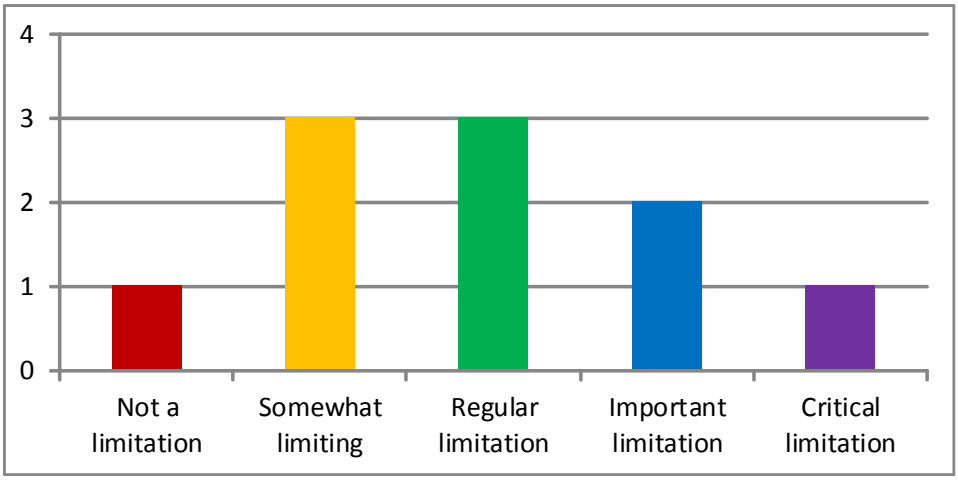


Figure C.13: Limitations: steep learning curve of personalization model

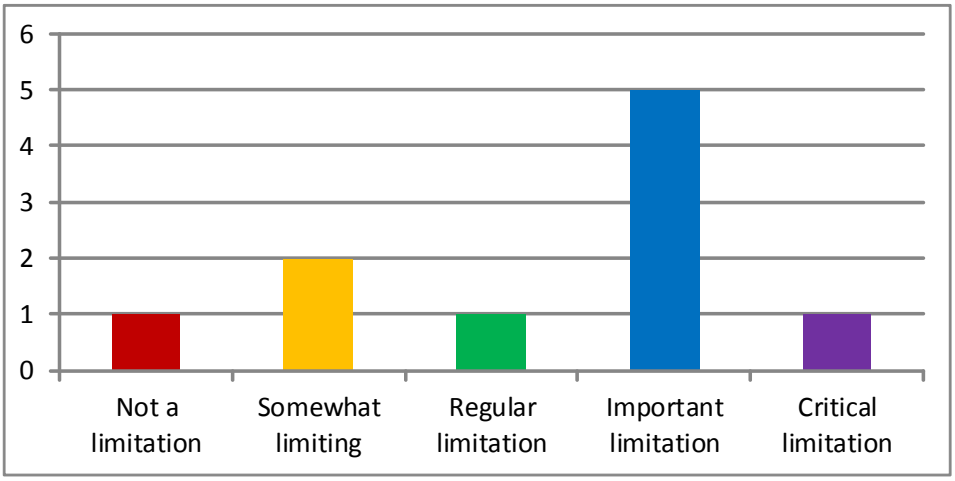


Figure C.14: Limitations: personalization model expressiveness

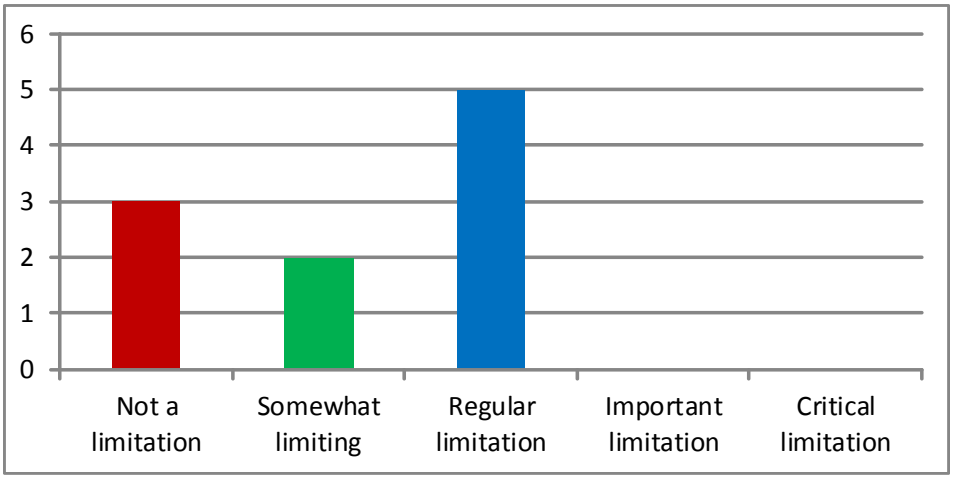


Figure C.15: Limitations: complex configuration process

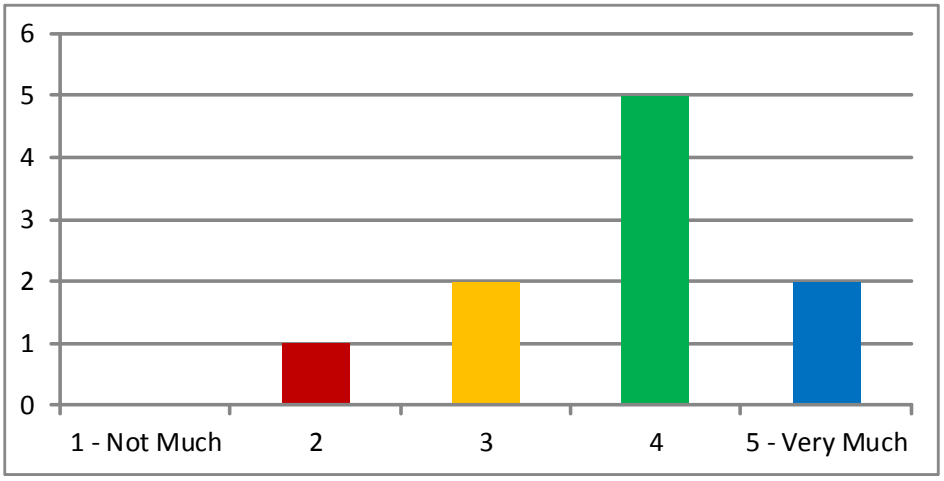


Figure C.16: How much would you be interested in using CAPE to provide personalization for your future applications?

C.3 Tests Applied to Other Applications - Results

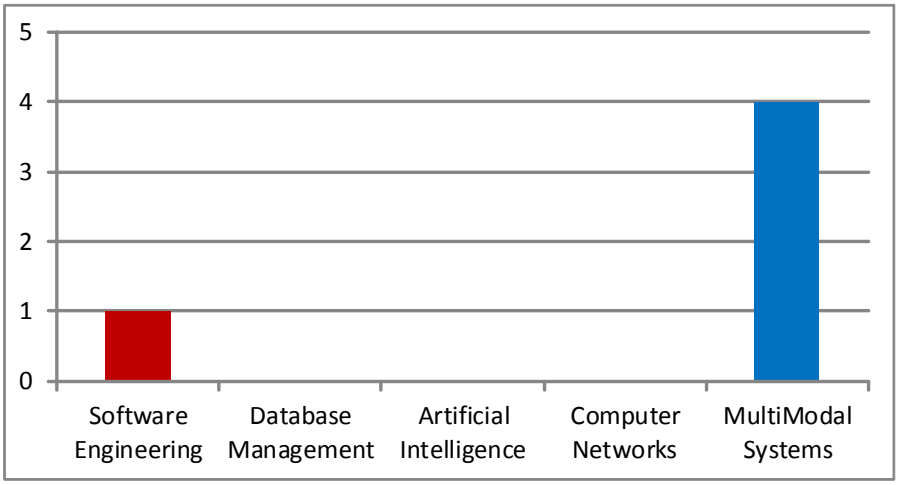


Figure C.17: What is your area of expertise?

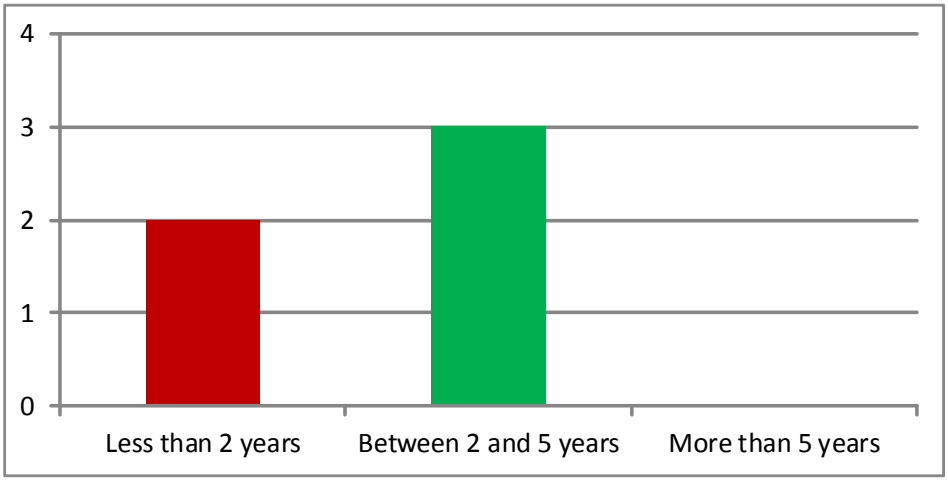


Figure C.18: For how long have you been working in that area?

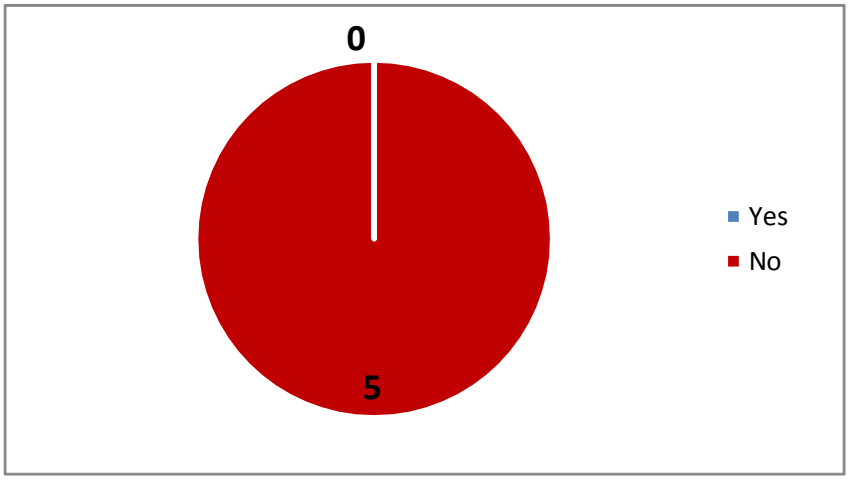


Figure C.19: Have you ever used any tools or approach to integrate personalization into any of your applications?

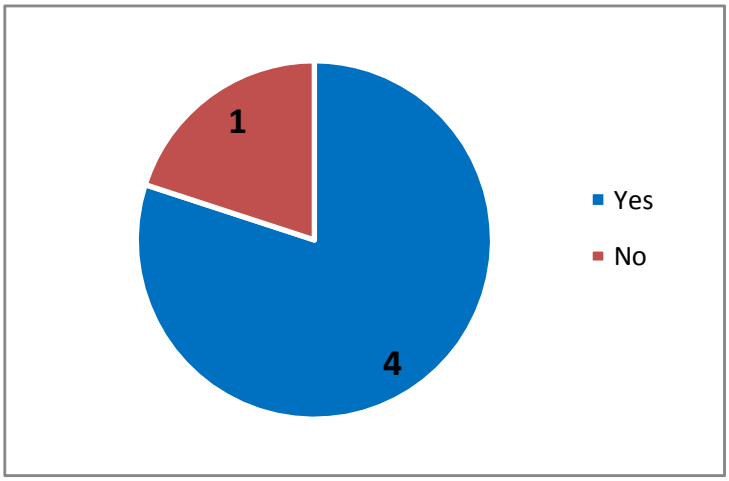


Figure C.20: Do you consider intuitive the connection between personalization - parameter - resource?

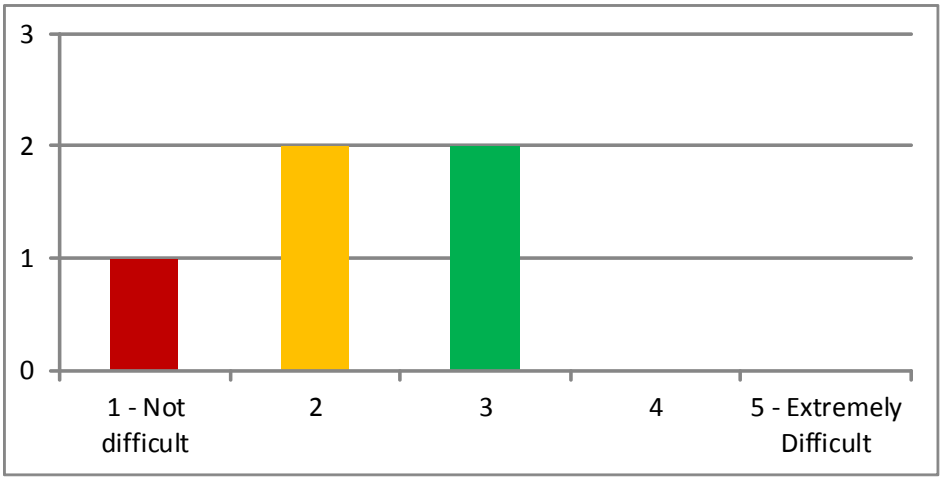


Figure C.21: How difficult was it to understand the configuration methodology for an application?

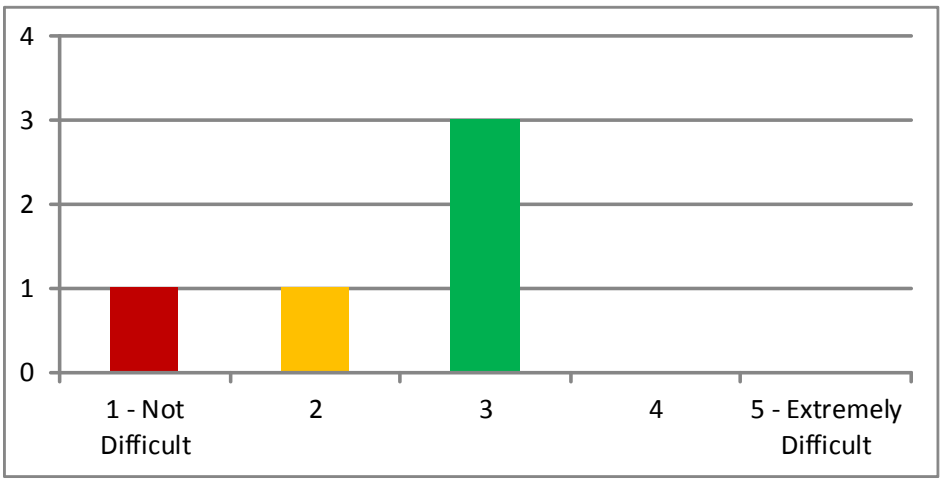


Figure C.22: How difficult was it to apply personalization to your application using CAPE methodology?

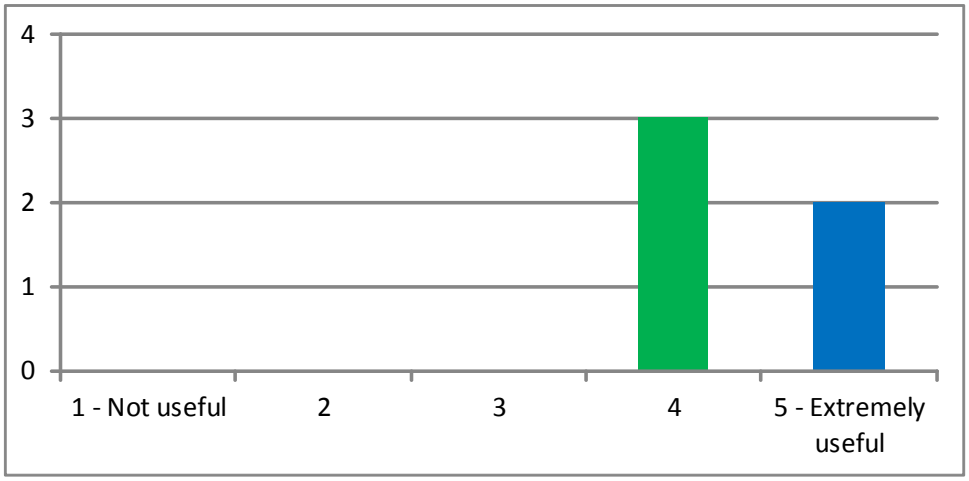


Figure C.23: How useful do you think is the notion of 'context', for personalization purposes?

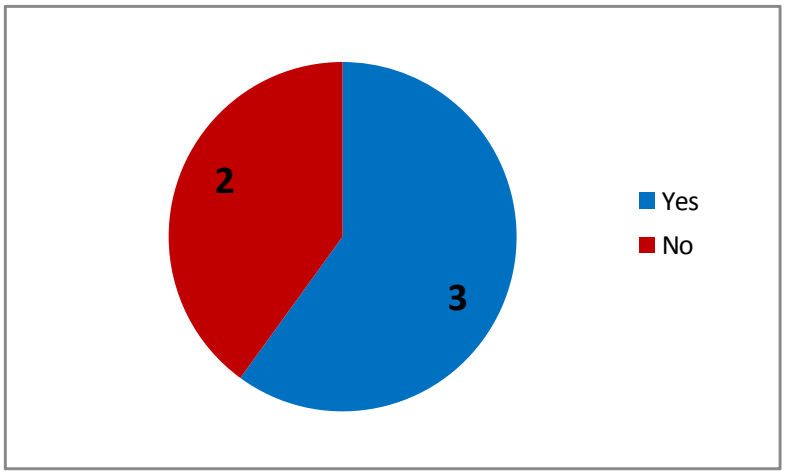


Figure C.24: Do you think there should be more types of external services?

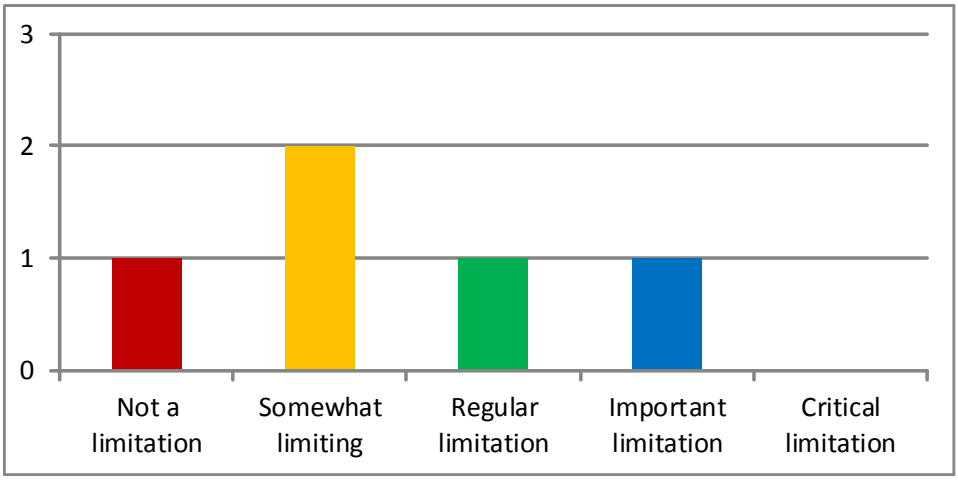


Figure C.25: Limitations: steep learning curve of personalization model

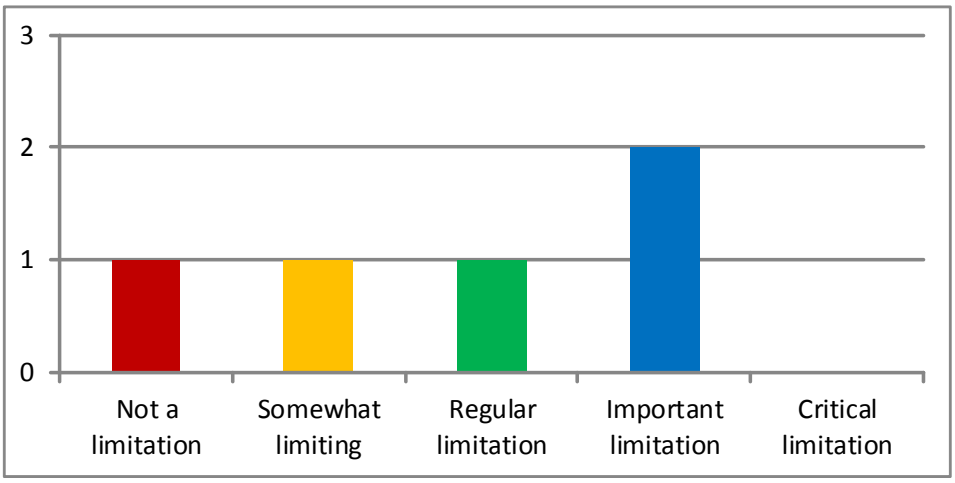


Figure C.26: Limitations: personalization model expressiveness

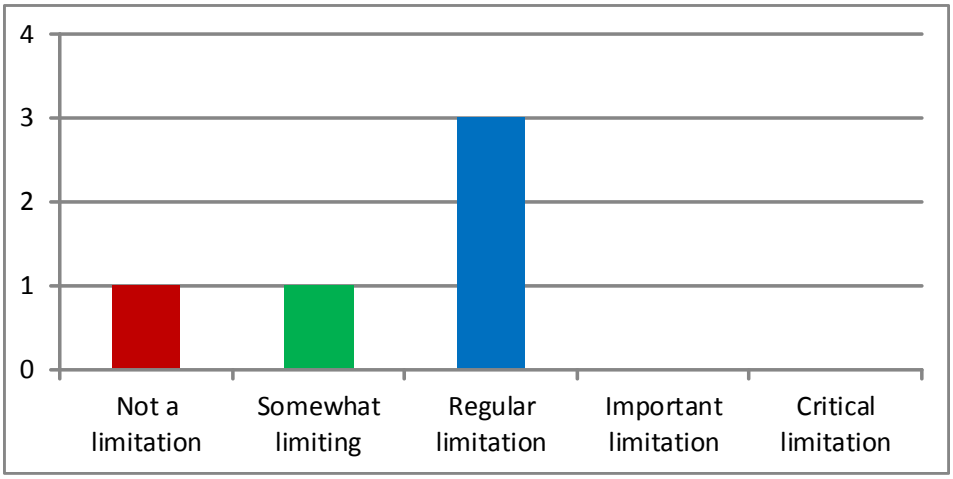


Figure C.27: Limitations: complex configuration process

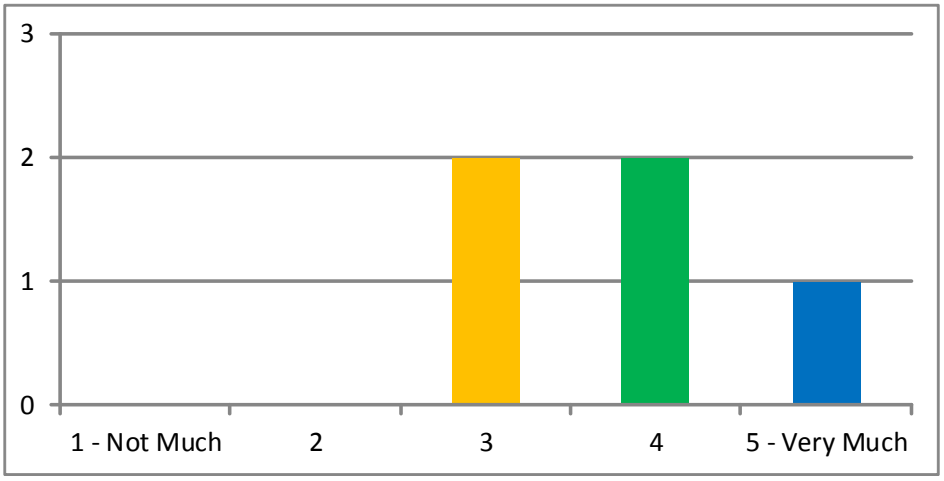


Figure C.28: How much would you be interested in using CAPE to provide personalization for your future applications?